České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Komprese konvolutorních vrstev neuronových sítí

Compression of convolutional layers in neural networks

Diplomová práce

Autor: **Bc. Teodor Kováč**

Vedoucí práce: **Ing. Petr Tichavský, DSc.**

Akademický rok: 2020/2021

Katedra: matematiky Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Teodor Kováč
Studijní program:	Aplikace přírodních věd
Studijní obor:	Matematická informatika
Název práce (česky):	Komprese konvolutorních vrstev neuronových sítí
Název práce (anglicky):	Compression of convolutional layers in neural networks

Pokyny pro vypracování:

1) Sepište všeobecný úvod do problematiky neuronových sítí a kanonického rozkladu tenzorů a vysvětlete algoritmy kanonických rozkladů při dané sensitivitě.
2) Pro již známé neuronové sítě , např. AlexNet, ResNet18 a Resnet50 nalezněte přibližné rozklady s různými hodnostmi (rank) a různými sensitivitami.
3) V neuronových sítích nahraďte konvolutorní vrstvy jejich rozklady. Zkoumejte, jak se mění jejich úspěšnost (chybovost) klasifikace v závislosti na hodnosti a sensitivitě rozkladů na databázích CIFAR 10 a CIFAR100.
4) Na základě vyhodnocení výše zmíněných experimentů navrhněte optimální strategii komprese konvolutorních vrstev neuronových sítí určených ke klasifikaci obrazů.

Doporučená literatura:

1) P. Tichavský, A. H. Phan, A. Cichocki, Sensitivity in tensor decomposition , IEEE Signal Processing Letters 26, 2019, 1653-1657

2) Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang, Model Compression and Acceleration for Deep Neural Networks. The principles, progress, and challenges. IEEE Signal Processing Magazine, January 2018, p. 126-136.

3) V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky, Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition, arXiv: 1412.6553. Published as a conference paper at ICLR 2015

4) F. Chollet, Deep Learning with Python, Manning publications 2018

Jméno a pracoviště vedoucího diplomové práce:

Ing. Petr Tichavský, DSc.
Ústav teorie informace a automatizace AV ČR, v.v.i., Pod Vodárenskou věží 4, Praha 8

Jméno a pracoviště konzultanta:

Datum zadání diplomové práce: 28.2.2020

Datum odevzdání diplomové práce: 6.1.2021

Doba platnosti zadání je dva roky od data zadání.

Poděkování:
Chtěl bych zde poděkovat především svému školiteli Ing. Petru Tichavskému, DSc. za pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé diplomové práce.

Čestné prohlášení:
Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 6. ledna 2021 Teodor Kováč

Název práce:

Komprese konvolutorních vrstev neuronových sítí

Autor: Teodor Kováč

Obor: Matematická informatika

Druh práce: Diplomová práce

Vedoucí práce: Ing. Petr Tichavský, DSc., Ústav teorie informace a automatizace AV ČR, v. v. i.

Abstrakt: Cílem této práce je zefektivnit funkčnost konvolučních neuronových sítí za použití kanonického rozkladu tenzorů. Za tímto účelem tato práce seznamuje čtenáře se základními pojmy a metodami konvolučních neuronových sítí a hledání tenzorových rozkladů. Teoretická část popisuje kompresi konvolučních vrstev. V praktické části je provedena implementace na sítích určených k úlohám strojového vidění.

Klíčová slova: Tenzor, Neuronové sítě, Kanonický rozklad, Konvoluční neuronové sítě, Konvoluční jádro, Levenberg-Marquardtův algoritmus, Krylovův podprostor.

Title:

Compression of convolutional layers in neural networks

Author: Teodor Kováč

Abstract: The aim of this thesis is to make convolutional neural networks more effective with the use of canonical polyadic tensor decomposition. Therefore this work introduces the reader to the basic principles of convolutional neural networks and shows some useful tensor decomposition algorithms. Mathematical description of the methods for the compression of convolutional layers may be found in the theoretical part of this thesis. These methods were implemented in the practical part and its results were evaluated using networks designed for image recognition tasks.

Key words: Tensor, Neural network, Canonical polyadic decomposition, Convolutional neural newtork, Convolutional kernel, Levenberg-Marquardt algorithm, Krylov subspace.

Obsah

Použité značení a zkratky

$\mathbb{N}$	množina přirozených čísel,
$\mathbb{R}$	množina reálných čísel,
vektory (tenzory 1.řádu)	značíme pomocí malých písmen a_1, a_2, b_1, b_2, x, y, atd.,
matice (tenzory 2.řádu)	značíme pomocí velkých písmen A, B, C, X, Y, atd.,
tenzory třetího a vyššího řádu (dále už jen tenzory)	značíme pomocí velkých písmen psaných kaligraficky $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{P}, \mathcal{T}$, atd.

Matice a vektory

Značení	**Význam**
$A \in \mathbb{R}^{n \times m}$	reálná matice o rozměrech $n \times m$, s prvky $A(i, j) \in \mathbb{R}$,
$A(i, :) \in \mathbb{R}^{n_2}$	i-tý řádek matice $A \in \mathbb{R}^{n_1 \times n_2}$,
$A(:, j) \in \mathbb{R}^{n_1}$	j-tý sloupec matice $A \in \mathbb{R}^{n_1 \times n_2}$,
A^T	transpozice matice A,
$rank(A)$	hodnost matice A definovaná jako počet lineárně nezávislých řádků, resp. sloupců matice A,
$\|\|x\|\| = (\sum_i x^2)^{\frac{1}{2}}$	Euklidovská norma vektoru,
$\|\|A\|\|_F = (\sum_i \sum_j A(i, j)^2)^{\frac{1}{2}}$	Frobeniova norma matice,
$1_K \in \mathbb{R}^K$	vektor ze samých 1 o rozměru $K \times 1$,
$\mathbb{I}_K$	jednotková matice o rozměrech $K \times K$,
$\{A_i\}_{i \in \hat{n}}$, kde $A_i \in \mathbb{R}^{I \times J}$	matice $[A_1, A_2, \ldots, A_n] \in \mathbb{R}^{I \times nJ}$.

Tenzory

Značení	**Význam**
$\mathcal{A}(i, j, k, l) \in \mathbb{R}$	(i, j, k, l) - tý prvek tenzoru $\mathcal{A}$,
$\text{vec}(\mathcal{A}) \in \mathbb{R}^{n_1 n_2 n_3 n_4}$	vektorizace tenzoru $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times n_4}$,

Úvod

Jedním z možných přístupů k řešení problémů umělé inteligence jsou *neuronové sítě*. Umělé neuronové sítě mají za vzor odpovídající struktury biologických neuronových sítí a jsou v počítačích uchovávany pomocí parametrů a vztahů těchto parametrů.

S potřebou zpracovávat neuronovými sítěmi čím dál tím větší množství dat roste i potřeba uchovávat stále rostoucí množství parametrů, což naráží na hardwareová omezení. V rámci této práce bychom se rádi zaměřili na využití *kanonického rozkladu tenzorů* při vývoji metod vedoucím k úsporám parametrů potřebných k uchovávání neuronových sítí.

V konvolučních neuronových sítích většinu parametrů tvoří váhy konvolučních vrstev. Tyto váhy bývají často reprezentovány jako čtyř-dimenzionální objekt, tenzor čtvrtého řádu. Jedním ze způsobů, jak ušetřit nároky na paměť, je využití kanonického rozkladu tenzorů, kterým jsme se již zabývali v bakalářské práci. Konkrétněji budeme konvoluční vrstvy sítí nahrazovat větším množstvím menších vrstev. Tento přístup si dává za cíl nalezení optimální rovnováhy mezi úsporami paměti a úspěšností sítě.

Text práce je strukturovaný následujícím způsobem. Po krátkém úvodu do terminologie operací s tenzory přichází kapitola vymezující pojmy neuronových sítí a konvolučních neuronových sítí. Následuje kapitola dávající do souvislosti konvoluční neuronové sítě a kanonický rozklad tenzorů. Na jednotlivá konvoluční jádra je nahlíženo jako na tenzory čtvrtého řádu a jsou nahrazována kanonickými rozklady, čímž dochází k úspoře parametrů. Popis a výsledky prvních experimentů s neuronovými sítěmi jsou uvedeny ve třetí kapitole. V této kapitole pracujeme s relativně malými sítěmi a výsledky slouží jako motivace pro aplikaci v reálně používaných sítích Res-Net a VGG. Čtvrtá kapitola popisuje tzv. *sensitivitu* kanonického rozkladu a zároveň čtenáře seznamuje s algoritmem uzpůsobeným k hledání rozkladů s nízkou hodnotou tohoto parametru. Tento algoritmus se později ukáže být jako velice podstatný. K hlavním experimentům této diplomové práce se dostáváme v páté kapitole. Na příkladu dvou již výše zmiňovaných konvolučních neuronových sítí byla provedena náhrada jejich konvolučních vrstev, čímž vznikly nové sítě s nižším počtem parametrů. K rozkladu jednotlivých konvolučních jader byly použity rozdílné rozkladové algoritmy a výsledky byly následně srovnány.

Kapitola 1

Úvod do kanonického rozkladu tenzorů

Definice matice

Matice bude v tomto textu představovat dvojrozměrné pole prvků z tělesa. V této práci si vystačíme s tělesem $\mathbb{R}$. Nechť máme matici $M \in \mathbb{R}^{I \times J}$, kde $I, J \in \mathbb{N}$. M můžeme zapsat jako

$$M = \begin{pmatrix} M(1,1) & M(1,2) & \dots & M(1,J) \\ M(2,1) & M(2,2) & \dots & M(2,J) \\ \vdots & \vdots & \ddots & \vdots \\ M(I,1) & M(I,2) & \dots & M(I,J) \end{pmatrix}.$$

Khatri-Rao produkt

Nechť $A \in \mathbb{R}^{n_1 \times n_3}$, $B \in \mathbb{R}^{n_2 \times n_3}$. Khatri-Rao produkt je definován tímto způsobem

$$A \odot B = \begin{pmatrix} A(1,1)B(:,1) & A(1,2)B(:,2) & \dots & A(1,n_3)B(:,n_3) \\ A(2,1)B(:,1) & A(2,2)B(:,2) & \dots & A(2,n_3)B(:,n_3) \\ \vdots & \vdots & \ddots & \vdots \\ A(n_1,1)B(:,1) & A(n_1,2)B(:,2) & \dots & A(n_1,n_3)B(:,n_3) \end{pmatrix} \in \mathbb{R}^{n_1 n_2 \times n_3}.$$

Hadamardův produkt

Nechť $A, B \in \mathbb{R}^{n_1 \times n_2}$, potom Hadamardův produkt má tvar

$$A * B = \begin{pmatrix} A(1,1)B(1,1) & A(1,2)B(1,2) & \dots & A(1,n_2)B(1,n_2) \\ A(2,1)B(2,1) & A(2,2)B(2,2) & \dots & A(2,n_2)B(2,n_2) \\ \vdots & \vdots & \ddots & \vdots \\ A(n_1,1)B(n_1,1) & A(n_1,2)B(n_1,2) & \dots & A(n_1,n_2)B(n_1,n_2) \end{pmatrix} \in \mathbb{R}^{n_1 \times n_2}.$$

Definice tenzoru

Pod pojmem tenzor rozumíme zobecnění pojmu matice, tj. jedná se o vícerozměrné pole prvků z tělesa. Nechť máme tenzor $\mathcal{T}$, který můžeme matematicky zapsat jako $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, kde $N \in \mathbb{N}$ a $I_1, I_2, \dots, I_N \in \mathbb{N}$.

Řád tenzoru

Mějme $N \in \mathbb{N}$ a $I_1, I_2, \ldots, I_N \in \mathbb{N}$ a nechť

$$\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}.$$

Potom N nazýváme řádem tenzoru $\mathcal{T}$.

Poznámka

Ve většině případů si v této práci vystačíme s tenzory do čtvrtého řádu, tj.

$$\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}.$$

Tenzor hodnosti 1

Nechť máme tenzor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ řádu N. Tenzor $\mathcal{T}$ má hodnost 1 právě tehdy, když existují vektory $u^1 \in \mathbb{R}^{I_1}, u^2 \in \mathbb{R}^{I_2}, \ldots, u^N \in \mathbb{R}^{I_N}$ tak, že každý element tenzoru $\mathcal{T}$ lze vyjádřit následujícím způsobem

$$\mathcal{T}(x_1, x_2, \ldots, x_N) = \prod_{n=1}^{N} u^n(x_n) \quad \text{,kde} \quad x_n \in \{1, \ldots, I_n\} \ \forall n \in \{1, \ldots, N\}.$$

Hodnost tenzoru

Hodnost tenzoru $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ je minimální počet tenzorů hodnosti 1 potřebných k zapsání všech prvků tenzoru $\mathcal{T}$ pomocí následující sumace

$$\mathcal{T}(x_1, x_2, \ldots, x_N) = \sum_{r=1}^{R} \prod_{n=1}^{N} U^n(x_n, r). \tag{1.1}$$

Hodnost tenzoru značíme $\text{rank}(\mathcal{T}) = R$.

Matrizace tenzoru

Mějme tenzor $\mathcal{T}$ o rozměrech $I_1 \times I_2 \times \ldots \times I_N$. Matrizací tenzoru $\mathcal{T}$ podle n-té dimenze vznikne matice o rozměrech $I_n \times (I_1 \cdot I_2 \cdot \ldots \cdot I_{n-1} \cdot I_{n+1} \cdot \ldots \cdot I_N)$, kterou budeme značit $\mathcal{T}_{(n)}$. Původní $(i_1, i_2, \ldots, i_N)$-tý prvek tenzoru $\mathcal{T}$ je v matrizovaném tenzoru $\mathcal{T}_{(n)}$ na pozici (i_n, j), kde

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k \quad \text{a} \quad J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m.$$

Tuto konvenci matrizace tenzoru jsme převzali z [1].

Kanonický rozklad tenzoru

Nechť máme tenzor $\mathcal{T}$ N-tého řádu o hodnosti R. Potom matice

$$\begin{aligned} U^1 &= [U^1(:,1), U^1(:,2), \ldots, U^1(:,R)] \in \mathbb{R}^{I_1 \times R}, \\ U^2 &= [U^2(:,1), U^2(:,2), \ldots, U^2(:,R)] \in \mathbb{R}^{I_2 \times R}, \\ &\vdots \\ U^N &= [U^N(:,1), U^N(:,2), \ldots, U^N(:,R)] \in \mathbb{R}^{I_N \times R} \end{aligned}$$

z předešlé definice (1.1) nazýváme **faktorové matice** a $[[U^1, U^2, \ldots, U^n]]$ značí **kanonický rozklad**. V některých případech budeme namísto názvu kanonický rozklad používat jen zkratku **CP rozklad** z anglického *canonical polyadic decomposition*.

Pro tenzory čtvrtého řádu budeme kanonický rozklad značit $\mathcal{T} = [[A, B, C, D]]$.

Poznámka

Můžeme si všimnout, že pokud by se nám podařilo najít přesný kanonický rozklad například tenzoru $\mathcal{T} = [[A, B, C, D]]$, tak namísto původních $I_1 I_2 I_3 I_4$ parametrů nám k udržení tenzoru v paměti stačí pouze $(I_1 + I_2 + I_3 + I_4)R$ hodnot. Toto je jedna z výhod kanonického rozkladu, kterou budeme v práci dále využívat.

Algoritmy hledání kanonických rozkladů

Tenzorová algebra má mnoho podobného s maticovou algebrou, ale zároveň několik podstatných rozdílů. Zatímco pro hledání hodnosti matice existuje deterministický algoritmus o konečném počtu kroků, tak problém hledání hodnosti tenzoru řádu tři a více je NP-složitá úloha [10]. Od toho se také odvíjí fakt, že při hledání kanonického rozkladu se často musíme spokojit jen s jeho aproximací. Hledání rozkladu můžeme popsat pomocí vícerozměrné účelové funkce, na kterou použijeme některý z k tomu vyvinutých optimalizačních alogoritmů. Algoritmy iterativně hledají minima účelových funkcí, bohužel ale neumí zaručit, že se bude jednat o minima globální a je proto třeba rozkladové algoritmy pouštět vícekrát s různými inicializacemi. Nejčastěji používáné algoritmy byly popsány v bakalářské práci [11], kde byly shrnuty i jednotlivé vlastnosti na příkladech tenzorů maticového násobení a tenzorů násobení polynomů.

Kapitola 2

Neuronové sítě

V této kapitole čtenáře uvedeme do tématiky neuronových sítí a zavedeme základní názvosloví, které budeme dále v textu používat. Názvosloví bylo převzato z [2] a [3]. Zároveň by kapitola měla sloužit jako motivace pro studium komprese konvolučních vrstev neuronových sítí za pomoci kanonického rozkladu.

2.1 Neuronové sítě

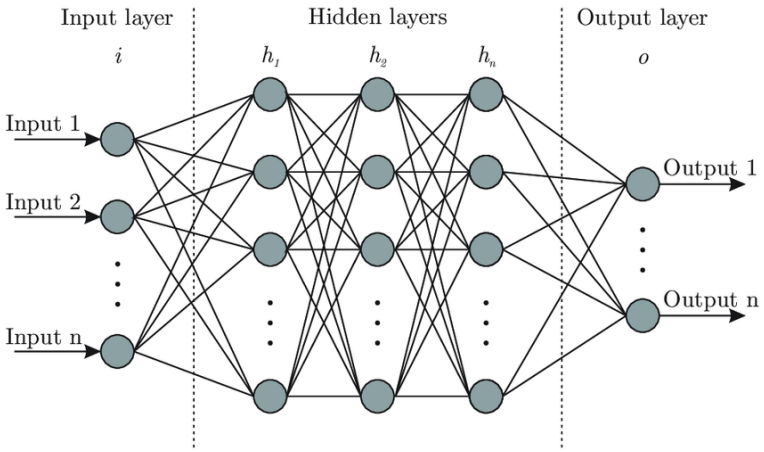
Umělá neuronová síť je matematický výpočetní model, který se snaží chováním připodobnit biologickým strukturám. Obor umělých neuronových sítí započala práce Warrena McCullocha a Waltera Pittse v roce 1943, ve které vytvořili jednoduchý model základního kamene neuronové soustavy - neuronu [8]. V jejich případě však parametry neuronu nabývaly jen hodnot z množiny {-1,0,1} a k zobecnění došlo až v roce 1957 Frankem Rosenblattem, který zavedl pojem *perceptron*. S využitím perceptronu Rosenblatt popsal první algoritmy určené k rozpoznávání objektů. Navíc se Franku Rosenblattovi společně s Charlesem Wightmanem podařilo sestrojit první počítač aplikující jednoduchou neuronovou síť k rozpoznávání obrazců. Počítač nesl název *Mark I Perceptron* a jako vstup pro rozpoznávání sloužila světelná tabule. Světelná tabule byla snímána polem 20×20 fotovodiči, které sloužily jako celkem 400 vstupních bodů neuronové sítě.

Další průlom v oboru umělých neuronových sítí přišel až v roce 1986, kdy byl publikován do dnes používaný algoritmus pro učení vícevrstvých neuronových sítí, tzv. algoritmus zpětného učení neboli *backpropagation algorithm*. Algoritmus zpětného učení společně s vývojem počítačů nastartoval zájem o další výzkum v tomto odvětví. V dnešní době neuronové sítě nabyly na popularitě a jejich výzkum vedl především v oboru rozpoznávání obrazu k mnoha přelomovým aplikacím. Jako například již v roce 1989 úspěšná implementace konvoluční neuronové sítě k rozpoznávání psaných číslic na poštovních dopisech [7] apod.

2.1.1 Matematický popis

Vstupní vrstva: Vstupní vrstva je jedno či vícerozměrné pole reálných čísel modelující vstupní informace daného problému. V případě úloh rozpoznávání obrazu se jedná o dvourozměrnou matici pro černobílý obraz a o tenzor třetího řádu pro obraz barevný.

Skrytá vrstva: Ve skryté vrstvě jsou prováděny nelineární transformace vstupních dat. Skládá se z různých struktur perceptronů a spojení mezi nimi.



Obrázek 2.1: *Grafické zobrazení obecné neuronové sítě [23].*

Perceptron: Po vzoru neuronů v biologických neuronových sítích, je perceptron základní výpočetní uzel umělé neuronové sítě. Obecně má perceptron N reálných vstupů.

Nechť máme na vstupu perceptronu reálný vektor $x = (x_1, x_2, \ldots, x_N) \in \mathbb{R}^N$. Jednotlivé vstupy - prvky vektoru x, jsou perceptronem ohodnoceny tzv. váhami, které označíme jako vektor $w = (w_1, w_2, \ldots, w_N) \in \mathbb{R}^N$. K ohodnoceným vstupům bývá často navíc přičítán *práh*, obvykle nazývaný *bias*. Výstupní hodnotu daného perceptronu modeluje *aktivační funkce*. Perceptron je jednoznačně určený vahami, prahem a aktivační funkcí. Jako příklad uvedeme výpočet výstupu i-tého perceptronu v j-té vrstvě

$$Y_{i,j} = f\left(\sum_{n=1}^{N} x_n w_n + b\right) = f(x^t w + b).$$

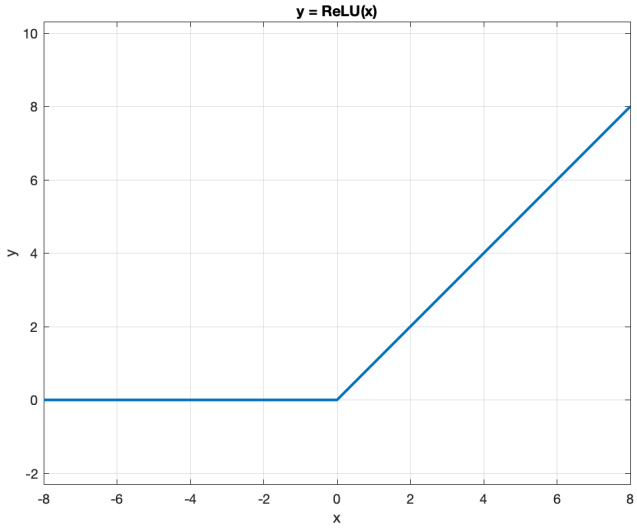
Poznámka

V této práci budeme jako aktivační funkci používat výhradně funkci *Rectified Linear Units - ReLu*, nebo-li pozitivní část argumentu viz. Obr. 2.2.

Výstupní vrstva: Jedná se o poslední vrstvu neuronové sítě, která na základě výstupu ze skryté vrstvy sděluje výsledek.

2.2 Učení neuronové sítě

Aby neuronové sítě mohli úspěšně řešit požadované úlohy, musí v první fázi dojít ke správné adaptaci jejich parametrů. V úlohách strojového vidění bývají pravidla pro volbu hodnot parametrů nejčastěji určována již zmiňovaným algoritmem zpětného učení. Zpětné učení spadá do kategorie učení s učitelem, které vyžaduje dostatečně velkou předem strukturovanou tréninkovou databázi uspořádaných dvojic - [Vstup do sítě, Požadovaný výstup]. Vstup do sítě slouží k výpočtu aktuálního výstupu, který je poté porovnáván s požadovaným výstupem a jejich rozdíl udává chybovou funkci [5]. Chybová funkce je poté minimalizována podle jednotlivých parametrů sítě.



Obrázek 2.2: *Aktivační funkce ReLu*

2.2.1 Problém přeučení sítě

V angličtině problém nazývaný *overfitting* týkající se správné volby velikosti neuronové sítě. V případě, když bychom zvolili nedostatečný počet perceptronů a vrstev neuronové sítě, tak by se mohlo stát, že struktura nebude dostatečně bohatá k řešení požadovaného problému. V praxi to znamená, že se v procesu učení algoritmus zastaví v mělkém lokálním minimu. Na druhé straně, pokud zvolíme síť přehnaně velkou, může docházet k takzvanému *overfittingu*, česky také přeučení [4]. Jedná se o jev, kdy síť přílišně zobecňuje tréninkové vzory včetně jejich nepřesností, tudíž pro vzory mimo tréninkovou databázi dává chybné výsledky.

2.3 Konvoluční neuronové sítě

Vhodným nástrojem pro řešení problémů strojového vidění, konkrétněji pro problémy rozpoznávání obrazu, se ukazují být *konvoluční neuronové sítě* [9]. Tyto sítě oproti běžným neuronovým sítím navíc obsahují takzvané *konvoluční vrstvy*, které provádějí operaci konvoluce určenou *konvolučním jádrem*. Strukturu konvoluční vrstvy a její funkčnost předvedeme na konkrétním příkladu průchodu dat touto vrstvou.

Mějme S vstupů do konvoluční vrstvy, kde každý vstup je matice o rozměrech $M \times M$. Konvoluční jádro pro každý vstupní kanál $\{1, 2, \ldots, S\}$ obsahuje T různých matic o rozměrech $N \times N$. Tyto matice nazýváme filtry a nebo stejně jako u perceptronu - váhy. Celkem tedy konvoluční vrstva $S \times T$ filtrů. Na výstupu budeme mít T matic, které vzniknou konvolucí vstupních matic s jednotlivými filtry. Matematicky zapsáno, nechť máme vstupní matice $X^1, X^2, \ldots, X^S$, které označíme jako tenzor třetího řádu

$$\mathcal{X} = [X^1, X^2, \ldots, X^S] \in \mathbb{R}^{M \times M \times S}.$$

Stejně tak zapíšeme všechny filtry daného konvolučního jádra pomocí tenzoru. V tomto případě se jedná o tenzor čtvrtého řádu

$$\mathcal{A} \in \mathbb{R}^{N \times N \times S \times T},$$

kde N je rozměr filtru, S počet vstupních kanálů a T počet výstupních kanálů. Konvolučních vrstev v konvolučních neuronových sítích bývá vícero, a proto budeme značit

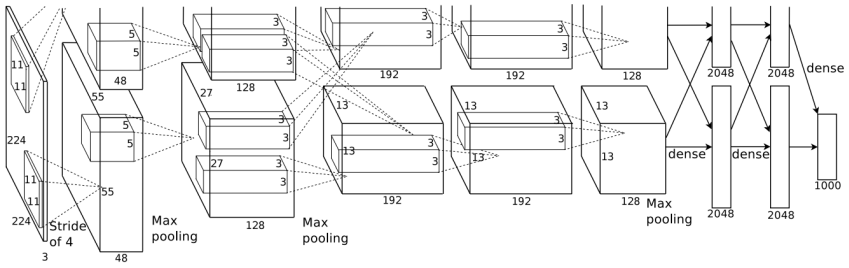
$$\mathcal{X}^{(K)} \dots \text{vstupní matice do } K\text{-té konvoluční vrstvy},$$

$$\mathcal{A}^{(K)} \dots \text{tenzor filtrů } K\text{-té konvoluční vrstvy}.$$

Nyní můžeme zapsat výpočet výstupu z K-té konvoluční vrstvy, tj. vstupu do $(K+1)$-ní vrstvy

$$X^{(K+1)}(x, y, t) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} \sum_{s=1}^{S} \mathcal{A}^{(K)}(i - x + \delta, j - y + \delta, s, t) X^{(K)}(i, j, s), \tag{2.1}$$

kde $\delta = (N-1)/2$. Tenzor filtrů bežně dosahuje velikosti několika milionů parametrů a stává se tím pádem jedním z nejobjemnějších objektů konvolučních neuronových sítí.



Obrázek 2.3: *Grafické zobrazení konvoluční neuronové sítě. Zdroj [24]*

Kapitola 3

Aproximace konvolučního jádra CP rozkladem

3.1 Odvození

Připomeňme si výpočet v K-té vrstvě konvoluční neuronové sítě.

$$X^{(K+1)}(x,y,t) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} \sum_{s=1}^{S} \mathcal{A}^{(K)}(i-x+\delta, j-y+\delta, s, t) X^{(K)}(i,j,s). \tag{3.1}$$

Cílem je snížit počet parametrů potřebných k popisu tenzoru filtrů $\mathcal{A}^{(k)} \in \mathbb{R}^{N\times N\times S\times T}$. Zde se nabízí využití CP rozkladu. Ačkoliv CP rozklad nemusí dokonale popisovat původní tenzor, tak jeho aproximace by mohla být dostatečná a nemusela by negativně ovlivnit funkčnost sítě.

Problém nyní popíšeme matematicky. Provedeme náhradu tenzoru filtrů $\mathcal{A}^{(K)}$ jeho kanonickým rozkladem o hodnosti R. Mějme $\mathcal{A}^{(K)} = [[A^X, A^Y, A^S, A^T]]$. Zapíšeme vyjádření (i,j,s,t)-tého prvku

$$\mathcal{A}^{(K)}(i-x+\delta, j-y+\delta, s, t) = \sum_{r=1}^{R} A^X(i-x+\delta, r) A^Y(j-y+\delta, r) A^S(s,r) A^T(t,r). \tag{3.2}$$

Nyní dosadíme (3.2) do (3.1) a provedeme úpravy sumací

$$X^{K+1}(x,y,t) = \sum_{r=1}^{R} A^T(t,r) \left(\sum_{i=x-\delta}^{x+\delta} A^X(i-x+\delta, r) \left(\sum_{j=y-\delta}^{y+\delta} A^Y(j-y+\delta, r) \left(\sum_{s=1}^{S} A^S(s,r) X^k(i,j,s) \right) \right) \right). \tag{3.3}$$

K udržení kanonického rozkladu v paměti je zapotřebí $(2N + S + T)R$ proměnných, což pro vhodnou volbu hodnosti R může být výrazně méně než N^2ST proměnných původního tenzoru.

Dále si můžeme všimnout, že se původní konvoluce dá zapsat pomocí čtyř menších konvolucí. Zavedeme si pomocné tenzory U^S, U^{SY}, U^{SYX}.

$$U^S(i,j,r) = \sum_{s=1}^{S} A^S(s,r) X^K(i,j,s), \tag{3.4}$$

$$U^{SY}(i,y,r) = \sum_{j=y-\delta}^{y+\delta} A^Y(j-y+\delta, r) U^S(i,j,r), \tag{3.5}$$

$$U^{SYX}(x,y,r) = \sum_{i=x-\delta}^{x+\delta} A^{X}(i-x+\delta,r)U^{SY}(i,y,r), \tag{3.6}$$

$$X^{K+1}(x,y,t) = \sum_{r=1}^{R} A^{T}(t,r)U^{SYX}(x,y,r). \tag{3.7}$$

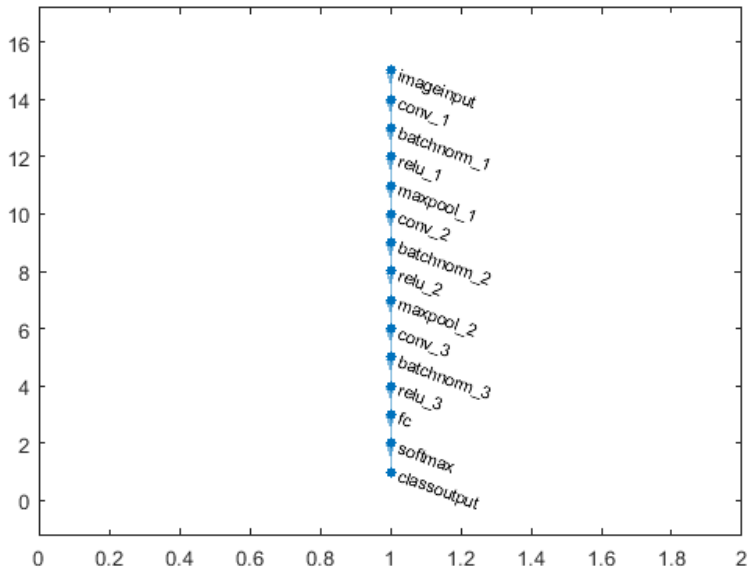
Tím dostáváme teoretický návod, jak rozdělit výpočet z jedné konvoluční vrstvy do čtyř hypoteticky menších vrstev.

3.2 Experiment s databází DigiDataset

První experiment této práce byl prováděn se základní konvoluční neuronovou sítí určenou k rozpoznávání číslic 0 až 9. Jedním cílem experimentu bylo ukázat, že náhradou původního konvolučního jádra jeho aproximací nedojde k podstatnému zhoršení přesnosti rozpoznávání sítě. Druhým cílem experimentu bylo odhadnutí vhodných hodností CP rozkladů. Experiment byl prováděn ve vývojovém prostředí MATLAB a sloužil jako úvod do práce s těmito objekty.

3.2.1 Popis CNN

Konvoluční neuronovou síť určenou k rozpoznávání číslic jsme sestavili tak, jak bylo popsáno v [12]. Síť má celkem 7 vrstev, z toho 3 vrstvy konvoluční. Vstupní vrstva má rozměry $28 \times 28 \times 1$, tj. přijímá černobílé obrázky o rozměrech 28×28. Trénování sítě probíhalo na databázi *DigiDataset* obsažené v knihovně MATLAB *Deep Learning Toolbox* [13]. Dataset obsahuje 10 000 obrázků.



Obrázek 3.1: *Architektura konvoluční neuronové sítě pro rozpoznávání čísel.*

3.2.2 Implementace

Databáze DigiDataset byla rozdělena do dvou částí. Náhodně vybraných 7 000 obrázků bylo použito pro trénování sítě a zbytek byl použit k následnému určení přesnosti. Síť byla příkazem *trainNetwork()* natrénována na rozpoznávací přesnost $97,8\overline{3}\%$.

Označíme si jádra konvolučních vrstev *conv_1*, *conv_2*, *conv_3* jako $\mathcal{A}_1, \mathcal{A}_2$ a $\mathcal{A}_3$. Jádro první konvoluční vrstvy $\mathcal{A}_1$ má jen malé rozměry $3\times3\times1\times8$ a proto vrstvu necháme nezměněnou. Zaměříme se na konvoluční jádra druhé a třetí konvoluční vrstvy, která nabývají podstatně větších rozměrů. Konkrétněji mají konvoluční jádra $\mathcal{A}_2$ a $\mathcal{A}_3$ rozměry $3 \times 3 \times 8 \times 16$ a $3 \times 3 \times 16 \times 32$.

Hlavní část experimentu spočívala v hledání kanonických rozkladů těchto dvou tenzorů konvolučních jader. Z faktorových matic jsme skládali aproximace tenzorů $\mathcal{A}_2$ a $\mathcal{A}_3$ a těmi jsme zaměňovali původní konvoluční jádra. Následně jsme vyhodnocovali rozpoznávací přesnost takto vzniklé sítě. Abychom získali lepší představu o vhodných hodnotách hodnosti CP aproximace, tak jsme experiment prováděli s větším množstvím různých parametrů hodnosti R a sledovali jsme závislost rozpoznávací přesnosti výsledné sítě.

K rozkladu tenzorů čtvrtého řádu jsme použili *nonlinear least squares* algoritmus z knihovny *Tensorlab 3.0* [14].

Uvedeme schéma experimentu pro příklad náhrady konvolučního jádra $\mathcal{A}_2$. V případě $\mathcal{A}_3$ se jedná o analogický postup.

1. Zvolíme hodnost R a provedeme tři[1] rozklady tenzoru vah konvoluční sítě pomocí *cpd_nls()*.
2. Vybereme faktorové matice $[[A^X, A^Y, A^S, A^T]]$ s nejmenší chybou rozkladu (error). Chybu počítáme jako
$$\text{error} = \|\mathcal{A} - [[A^X, A^Y, A^S, A^T]]\|_F^2$$
3. Aproximaci původního tenzoru $\mathcal{A}_2$ složenou z faktorových matic A^X, A^Y, A^S, A^T označíme $\mathcal{Y}_2$.
4. Váhy příslušné konvoluční vrstvy zaměníme příslušnou aproximací $\mathcal{Y}_2$.
5. Napočítáme, jak se změnila přesnost rozpoznávání na stejné sadě testovacích vzorků.

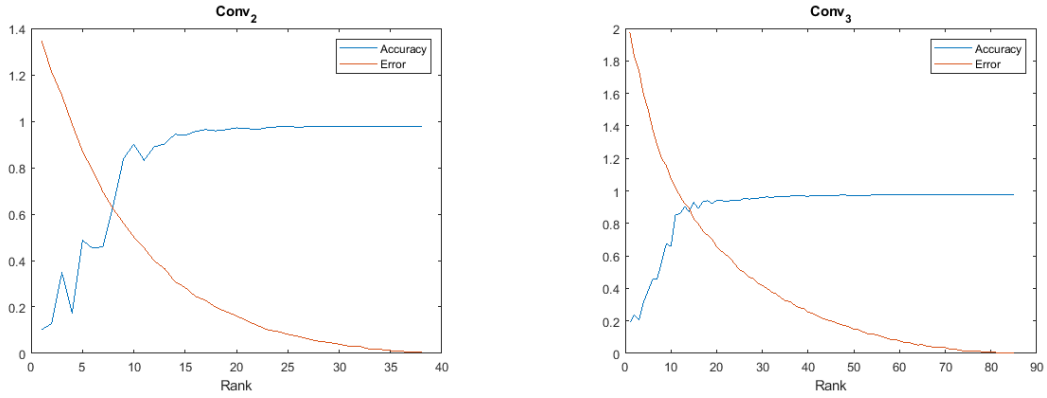
3.2.3 Výsledky

Pro tenzor $\mathcal{A}_2$ jsme za hodnotu hodnosti pro hledání CP rozkladu na počátku zvolili $R = 38$. V dalších opakováních jsme hodnost postupně snižovali. Pro všechny hodnosti vyšší než 38 jsou faktorové matice objemnější než původní tenzor a tudíž není důvod se jimi zabývat. Výsledky jsou shrnuty v grafu 3.2 (a), ze kterého můžeme vidět, že všechny aproximace CP rozkladem s hodností větší než 15 udržely přesnost rozpoznávání sítě nad 95%. Až u hodnoty 15 a nižší začala přesnost rozpoznávání výrazněji klesat. K udržení CP rozkladu tenzoru $\mathcal{A}_2$ o hodnosti $R = 16$ je zapotřebí 480 proměnných. K zapsání původního tenzoru bylo třeba celkem $3\times3\times8\times16 = 1152$ proměnných. Jedná se tedy o úsporu necelých 60% nároků na paměť.

V případě druhého tenzoru $\mathcal{A}_3$ jsme začínali s hodností $R = 85$. I při nahrazení tenzoru jeho CP aproximací s hodností 26 se úspěšnost sítě stále držela nad 95%. Namísto původních $3\times3\times16\times32 = 4608$ proměnných nám k popisu faktorových matic o hodnosti 26 stačí pouze 1404 proměnných. Jedná se o úsporu 70% nároků na paměť. Výsledky přikládáme vyobrazeny formou grafu 3.2 (b).

Kompletní výstup tohoto experimentu je k nalezení v příloze A.

[1] Jelikož je prvotní inicializace náhodná a prostor obsahuje velké množství lokálních minim, tak počítáme aproximaci třikrát a poté vybereme tu s nejmenší chybou rozkladu.



(a) (b)

Obrázek 3.2: Grafy závislosti přesnosti na hodnosti R rozkladu

3.3 Experiment se sítí AlexNet

Předešlý experiment na jednoduché síti pro rozpoznávání čísel ukázal, že náhrady konvolučních jader jejich CP rozkladem by měly vést k výraznému snížení nároků na paměť. Použili jsme stejný postup a experiment jsme provedli s reálně používanou sítí AlexNet.

3.3.1 Popis sítě

Síť AlexNet je určena pro rozpoznávání obrazu a byla poprvé publikována Alexem Krizhevskym v roce 2012. Síť obsahuje 60 milionů parametrů, 5 konvolučních vrstev a její vstup má rozměry 224×224×3, tj. RGB[2] obrázek o rozměrech 224 × 224 pixelů. Přikládáme grafické znázornění struktury AlexNet viz. 3.3.

Síť je volně dostupná předtrénovaná na databázi *ImageNet* [21]. Pro náš účel jsme zvolili databázi menší, konkrétně *Caltech 101* [15], která obsahuje více než 8 000 obrázků rozdělených do 101 kategorií. Pro větší rozpoznávací přesnost bylo třeba s databází Caltech 101 provést tzv. *transfer learning*. Transfer learning používáme v případě, kdy síť má již klíčové prvky natrénované a je jen potřeba je správně "roztřídit"do jednotlivých kategorií. Schéma transfer learningu:

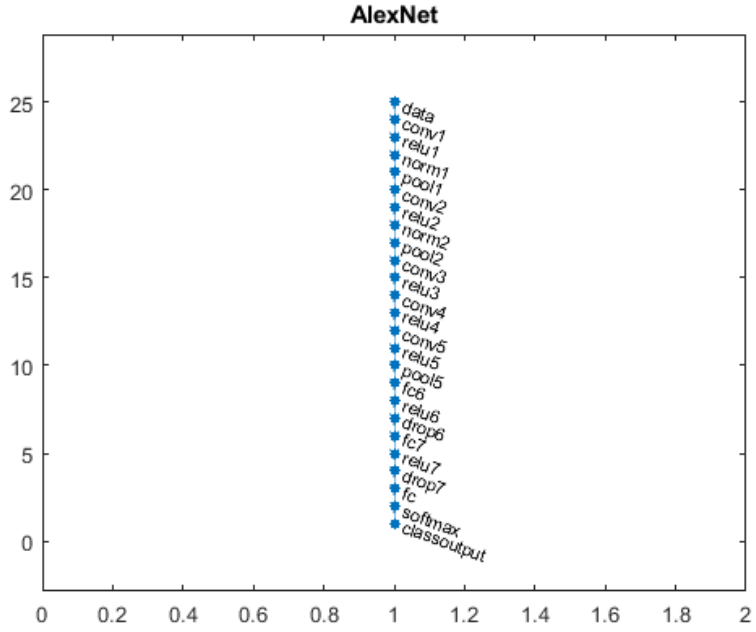
1. Z AlexNet odstraníme poslední tři vrstvy, tj. *fc* - fully connected, *softmax*, *classoutput*.
2. Na konec sítě napojíme nové nenatrénované vrstvy stejného typu.
3. Provedeme učení, při kterém budeme adaptovat jen parametry posledních tří vrstev. Databázi jsme opět rozdělili, trénink probíhá s náhodně vybranými 70% obrázků.

Validace proběhla na zbylých 30% obrázků. Tímto postupem úspěšnost rozpoznávání sítě dosáhla hodnoty 95,95%.

3.3.2 Implemetace a výsledky

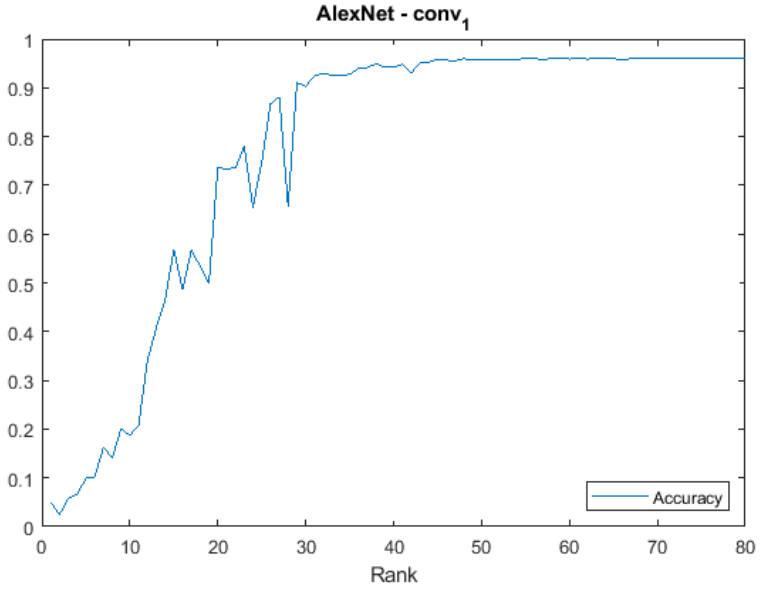
Experiment s AlexNet byl analogií experimentu popsaného v 3.2.2. Postup jsme aplikovali hned na první konvoluční vrstvu sítě, která má rozměry $11 \times 11 \times 3 \times 96$, což odpovídá 34 848 proměnným.

[2]Barevný model založený na aditivním míchání červené, zelené a modré barvy.



Obrázek 3.3: *Schematický graf konvoluční neuronové AlexNet.*

V průběhu experimentu jsme zjistili, že nahrazení konvolučního jádra jeho aproximací CP rozkladem s hodností 80 a větší nijak neovlivní přesnost rozpoznávání. Tudíž uvádíme výsledky jen s hodnostmi 80 a nižšími, viz graf 3.4. Můžeme si všimnout, že úspěšnost rozpoznávání se drží nad 95% i při aproximaci rozkladem s hodností $R = 43$. V takovém případě k uchování aproximace tenzoru postačuje pouhých 5203 proměnných, což by znamenalo úsporu 85% nároků na paměť. Kompletní výsledky jsou k nalezení v příloze A.



Obrázek 3.4: *Graf závislosti přesnosti na hodnosti rozkladu 1. konvoluční vrstvy AlexNet.*

Kapitola 4

Sensitivita a algoritmus KLM

Experimenty v předešlé kapitole měly sloužit především jako motivace pro implementaci CP rozkladu do konvolučních vrstev. Nyní bychom se rádi zaměřili na konkrétní kompresi konvolučních vrstev a náhrady větších konvolučních vrstev vrstvami řádově menšími, tak jak bylo popsáno v 3.4. Tím bychom měli dosáhnout reálné úspory nároků na paměť a snížení počtu proměnných.

Rádi bychom tedy konvoluční jádra nahrazovali jejich CP rozkladem a zároveň je dále trénovali. Bude tedy docházet k adaptaci proměnných a proto by bylo dobré, aby rozklady byly stabilní vůči malým změnám, tj. aby při změně parametrů faktorových matic, způsobené například trénováním sítě, docházelo k úměrně velké změně rozkladu. Proto, než začneme vrstvy nahrazovat, tak v této kapitole představíme kritérium *sensitivity rozkladu*, které bylo zavedeno v [19]. Abychom toto kritérium mohli ihned využít, tak zde následně popíšeme *Krylov-Levenberg-Marquardtův algoritmus* (KLM) přímo vyvinutý k hledání rozkladů s nízkou sensitivitou.

4.1 Sensitivita

Mějme tenzor čtvrtého řádu $\mathcal{T}$ a k němu příslušící CP rozklad $\mathcal{T} = [[A, B, C, D]]$. Sensitivitou rozkladu budeme rozumět

$$s(A, B, C, D) = \lim_{\sigma^2 \to 0} \frac{1}{\sigma^2} E\left(||\mathcal{T} - [[A + dA, B + dB, C + dC, D + dD]]||_F^2\right),$$

kde

dA, dB, dC, dD	...	náhodné matice s nezávisle rozdělenými prvky s Gaussovým rozdělením nulovou střední hodnotou a variancí σ^2,
$E()$	...	operátor střední hodnoty,
σ^2	...	variance.

4.2 Algoritmy pro hledání kanonických rozkladů

Cílem této sekce je popsat Krylov-Levenberg-Marquardtův algoritmus s podmínkou na CP rozklady s omezenou sensitivitou [16]. Za tímto účelem čtenáři popíšeme nejprve klasický Levenberg-Marquardtův algoritmus, poté přidáme podmínku omezující sensitivitu a nakonec popíšeme výpočet pomocí Krylovova podprostoru.

4.2.1 Levenberg-Marquardtův algoritmus

Mějme $\mathcal{T} \in \mathbb{R}^{I\times J\times K\times L}$ a cílem je nalézt faktorové matice $[[A, B, C, D]]$. Faktorové matice označíme výrazem

$$\theta_A = \text{vec}(A),\ \ \theta_B = \text{vec}(B),\ \ \theta_C = \text{vec}(C),\ \ \theta_D = \text{vec}(D)$$
$$\theta = [\theta_A^T; \theta_B^T; \theta_C^T; \theta_D^T].$$

Dále si označíme účelovou funkci

$$\varphi(\theta) = \frac{1}{2}\|\text{vec}(\mathcal{T}) - \text{vec}(\mathcal{T}(\theta))\|^2.$$

Levenberg-Marquardtův algoritmus spočívá v sekvenci iterací

$$\theta \leftarrow \theta' = \theta - (H + \mu\mathbb{I})^{-1} g,$$

kde vektor g nazýváme chybovým gradientem a matici H Hessovou maticí. θ je dosavadní nejlepší odhad. Hessova matice je definována pomocí Jakobiho matice následujícím způsobem

$$\begin{aligned} J &= \frac{\partial \text{vec}\mathcal{T}}{\partial\theta}, \\ g &= J^T\varphi(\theta), \\ H &= J^T J. \end{aligned} \tag{4.1}$$

μ je tlumící parametr, který se průběžně mění podle toho, jak klesá účelová funkce. Popis pravidla pro volbu parametru μ v jednotlivých iteracích společně s explicitním výrazem výpočtu Jakobiho matice J lze nalézt v příloze C a B.

Nyní můžeme zapsat pseudokód Levenberg-Marquardtova algoritmu.

Pseudokód - Levenberg-Marquardtův algoritmus

Data: tenzor $\mathcal{T}$, předpokládaná hodnost, počet iterací, $[[A, B, C, D]]$ = náhodná inicializace
Result: $[[A, B, C, D]]$, chyba
$it = 0$;
$A_1 = A$;
$B_1 = B$;
$C_1 = C$;
$D_1 = D$;
while *(it < počet iterací) || (err > 1e-16)* **do**
  $H = \mathbb{J}^T\mathbb{J}$, $g = \mathbb{J}^T\varphi(\theta_{it})$;
  $\theta_{it+1} \leftarrow \theta_{it} - (H + \mu\mathbb{I})^{-1} g$;
  chyba = $\varphi(\theta_{it+1})$;
  přepočet parametru μ ;
  $it = it + 1$;
end
$A = A_{it+1}$;
$B = B_{it+1}$;
$C = C_{it+1}$;
$D = D_{it+1}$;

4.2.2 Levenberg-Marquardtův algoritmus s omezením na sensitivitu rozkladu

Nyní popíšeme vylepšení algoritmu Leveneberg Marquardt, které bude zaručovat CP rozklady s omezeným parametrem sensitivity. Vylepšení bylo převzato z [16].

Hledáme faktorové matice $[[A, B, C, D]]$ za podmínky $c(\theta) = c_0$, kde $c(\theta) = s(A, B, C, D)$. Levenberg-Marquardtův algoritmus proto modifikujeme následujícím způsobem

$$\theta_1^{'} = \theta_0 - (H + \mu\mathbb{I})^{-1}g + \frac{u_0^T(H + \mu\mathbb{I})^{-1}g}{u_0^T(H + \mu\mathbb{I})^{-1}u_0}(H + \mu\mathbb{I})^{-1}u_0,$$

kde

$$u_0 = \frac{\partial c(\theta)}{\partial \theta}\Big|_{\theta=\theta_0}.$$

Výsledné θ_1 iteračního cyklu bude získáno už jen správným přeškálováním, tj. $\theta_1 = \alpha\theta_1^{'}$, kde $\alpha = (c_0/c(\theta_1^{'}))^{1/4}$ tak, aby byla splněna podmínka $c(\theta_1) = c_0$. Opět můžeme algoritmus zapsat formou pseudokódu.

Pseudokód - Levenberg-Marquardtův algoritmus s omezením sensitivity

Data: tenzor $\mathcal{T}$, předpokládaná hodnost, požadovaná sensitivita c_0, počet iterací, $[[A, B, C, D]]$ = náhodná inicializace
Result: $[[A, B, C, D]]$, chyba
$it = 0$;
$A_1 = A$;
$B_1 = B$;
$C_1 = C$;
$D_1 = D$;
while *(it < počet iterací) || (err > 1e-16)* **do**
  $H = \mathbb{J}^T\mathbb{J}$, $g = \mathbb{J}^T\varphi(\theta_{it})$;
  $\theta_{it+1}^{'} \leftarrow \theta_{it} - (H + \mu\mathbb{I})^{-1}g + \frac{u_0^T(H+\mu\mathbb{I})^{-1}g}{u_0^T(H+\mu\mathbb{I})^{-1}u_0}(H + \mu\mathbb{I})^{-1}u_0$;
  $\theta_{it+1} = \alpha\theta_{it+1}^{'}$, kde $\alpha = (c_0/c(\theta_{it+1}^{'}))^{1/4}$;
  chyba = $\varphi(\theta_{it+1})$;
  přepočet parametru μ ;
  $it = it + 1$;
end
$A = A_{it+1}$;
$B = B_{it+1}$;
$C = C_{it+1}$;
$D = D_{it+1}$;

4.2.3 Technika Krylovova podprostoru

Zaměříme se na výpočet inverze $(H + \mu I)^{-1}g$ tak jak bylo popsáno v [16]. H je symetrická pozitivně definitní matice a g je chybový gradient. V této sekci předpokládáme, že výpočet produktu $y = Hx$, kde x je vektor příslušné velikosti, umíme vypočítat relativně rychle a bez nutnosti mít matici H uloženou přímo v paměti počítače. Postup výpočtu, který opodstatňuje tento předpoklad, bude popsán v sekci 4.2.4.

Inverze výrazu $(H + \mu I)$ bývá jedna z výpočetně nejnáročnějších částí celého algoritmu. Abychom tento úkon zjednodušili, nabízí se $N \times N$ rozměrnou matici nahradit následující aproximací

$$H \approx UWU^T,$$

kde U je matice o rozměrech $N \times M$ a W je symetrická matice $M \times M$. Parametr M předpokládáme malý a nazýváme ho hodností komplexity aproximace. Dále použijeme Woodburyho identitu [1] a dosadíme

$$(H + \mu I)^{-1} g \approx \frac{1}{\mu} g - \frac{1}{\mu} U(\mu W^{-1} + U^T U)^{-1}(U^T g). \tag{4.2}$$

Výraz na pravé straně vyžaduje inverzi dvou matic o rozměrech $M \times M$, což pokládáme za výhodnější než inverzi výrazu o rozměrech $N \times N$ na straně levé.

Otázkou zůstává, jak volit matic U. Matici U budeme volit jako ortogonální bázi Krylovova podprostoru generovaného

$$[g, Hg, H^2 g, \ldots, H^{M-1} g]$$

Volba prvního vektoru báze může být libovolná a jedinou podmínkou je, aby vektor nebyl ortogonální vůči vektoru příslušejícímu největšímu vlastnímu číslu. V našem případě jako první vektor volíme chybový gradient g a zbytek matice $U = [u_1, \ldots, u_M]$ dopočítáme Gram-Schmidtovou ortogonalizační metodou.

$$u_1 = \frac{g}{\|g\|}$$

$$u'_{i+1} = Hu_i - \sum_{j=1}^{i} u_j^T (Hu_i) u_j$$

$$u_{i+1} = \frac{u'_{i+1}}{\|u'_{i+1}\|}$$

A konečně, jelikož je U ortogonální matice, tak matici W získáme jako $W = U^T HU$.

4.2.4 Výpočet y = Hx

Matice H může dosahovat velkých rozměrů a proto by bylo vhodné ji nemuset uchovávat v paměti celou. V předešlé sekci jsme matici H využívali především při výpočtu ortogonální báze Krylovova podprostoru a matici jsme násobili příslušným vektorem. Budeme tedy chtít najít předpis pro výraz $y = Hx = J^T Jx$, kde

$$J = [J_A, J_B, J_C, J_D] = \left[\frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}A}, \frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}B}, \frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}C}, \frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}D}\right]$$

Nechť máme daný vektor x, který si pro přehlednost označíme následujícím způsobem

$$x = [\mathrm{vec}X_A; \mathrm{vec}X_B; \mathrm{vec}X_C; \mathrm{vec}X_D],$$

kde X_A, X_B, X_C a X_D mají příslušné rozměry jako faktorové matice A, B, C a D. Za využití vzorců pro výpočet Jakobiho matice z přílohy B si podrobněji rozepíšeme součin Jx

$$\begin{aligned} Jx &= J_A \mathrm{vec}X_A + J_B \mathrm{vec}X_B + J_C \mathrm{vec}X_C + J_D \mathrm{vec}X_D \\ &= \mathrm{vec}[[X_A, B, C, D]] + \mathrm{vec}[[A, X_B, C, D]] + \mathrm{vec}[[A, B, X_C, D]] + \mathrm{vec}[[A, B, C, X_D]] \\ &= \mathrm{vec}\mathcal{Z}, \end{aligned} \tag{4.3}$$

[1] $(UCV + A)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$

kde

$$\mathcal{Z} = [[X_A, B, C, D]] + [[A, X_B, C, D]] + [[A, B, X_C, D]] + [[A, B, C, X_D]].$$

Výpočet celého součinu $J^T Jx$ nyní můžeme rozdělit do dílčích částí.

$$\begin{aligned} J_A^T Jx &= \{[[\mathbb{I}_R, B, C, D]]_{(1)} \otimes \mathbb{I}_I\}\mathrm{vec}\mathcal{Z}, \\ &= \{[[\mathbb{I}_R, B, C, D]]_{(1)} \otimes \mathbb{I}_I\}\mathrm{vec}\mathcal{Z}_{(1)}, \\ &= \mathrm{vec}\{\mathcal{Z}_{(1)}[[\mathbb{I}_R, B, C, D]]_{(1)}^T\}. \end{aligned} \tag{4.4}$$

A analogické vzorce dostáváme i pro zbylé součiny

$$\begin{aligned} J_B^T Jx &= \mathrm{vec}\{\mathcal{Z}_{(2)}[[A, \mathbb{I}_J, C, D]]_{(2)}^{(T)}\}, \\ J_C^T Jx &= \mathrm{vec}\{\mathcal{Z}_{(3)}[[A, B, \mathbb{I}_K, D]]_{(3)}^{(T)}\}, \\ J_D^T Jx &= \mathrm{vec}\{\mathcal{Z}_{(4)}[[A, B, C, \mathbb{I}_L]]_{(4)}^{(T)}\}. \end{aligned} \tag{4.5}$$

Výsledný vektor y vypočítáme nyní snadno jako $y = [J_A^T; J_B^T; J_C^T; J_D^T]Jx$.

4.2.5 Výpočet chybového gradientu g

Chybový gradient $g = J^T \varphi(\theta_0)$ by bylo také užitečné v jednotlivých iteracích napočítávat bez nutnosti zapisování do paměti, což je možné dle následujících vzorců. Nejprve označíme

$$\mathcal{E} = \mathcal{T} - [[A, B, C, D]] \quad \text{a} \quad g = [g_A; g_B; g_C; g_D].$$

Poté můžeme jednoduše jednotlivé části chybového gradientu popsat

$$\begin{aligned} g_A &= \mathrm{vec}\{\mathcal{E}_{(1)}[[\mathbb{I}_R, B, C, D]]_{(1)}^T\}, \\ g_B &= \mathrm{vec}\{\mathcal{E}_{(2)}[[A, \mathbb{I}_R, C, D]]_{(2)}^T\}, \\ g_C &= \mathrm{vec}\{\mathcal{E}_{(3)}[[A, B, \mathbb{I}_R, D]]_{(3)}^T\}, \\ g_D &= \mathrm{vec}\{\mathcal{E}_{(4)}[[A, B, C, \mathbb{I}_R]]_{(4)}^T\}. \end{aligned} \tag{4.6}$$

Kapitola 5

Experimenty

V této kapitole uvedeme experimenty komprese konvolučních vrstev za účelem optimalizace neuronových sítí. Zároveň se podrobněji zaměříme na parametr sensitivity CP rozkladu a budeme sledovat jeho vliv na úspěšnost metody.

5.1 Příprava

Některé úkony spojené s náhradou konvolučních vrstev a CP rozkladem jsou z hlediska výpočetního času velice náročné a proto byly prováděny ve výpočetním centru *Ústavu Teorie Informace a Automatizace AV ČR*. Konkrétně byla tato část diplomové práce prováděna na procesoru *AMD Ryzen Threadripper 2990WX* v konfiguraci 32 jader, 128GB RAM a s grafickou kartou NVIDIA GeForce RTX 2080 Ti, kde algoritmy pro hledání CP rozkladů byly prováděny na procesoru a učení neuronových sítí probíhalo na grafické kartě. Vše bylo implementováno ve vývojovém prostředí *MATLAB*.

Základem všech experimentů je vždy konvoluční neuronová síť. V této práci jsme se zaměřili na sítě *Res-Net 18* a *VGG 16*. Sítě jsou volně dostupné předtrénované v prostředí Matlab. Abychom byli schopni měřit úspěšnost sítí, bylo opět potřeba provést tzv. Transfer Learning s některou z dostupných databází. Zvolili jsme databázi *ImageNet Large Scale Visual Recognition Challenge - ILSVRC12* [21]. Z časových důvodů jsme databázi omezili na jednu čtvrtinu jejího původního objemu. Bez omezení databáze ILSVRC12 obsahuje 1 261 406 obrázků rozdělených do 1 000 kategorií. My jsme v této práci použili jen prvních 250 tříd. Takto vzniklou databázi jsme rozdělili v poměru 70/30, kde větší část byla použita pro učení a menší část pro vyhodnocení úspěšnosti sítě. Každá jednotlivá kategorie poskytuje 668 až 3047 obrázků, které jsou popsány lexikografickou databází *WordNet* [22]. Několik vzorových obrázků je k nalezeni v příloze **??**.

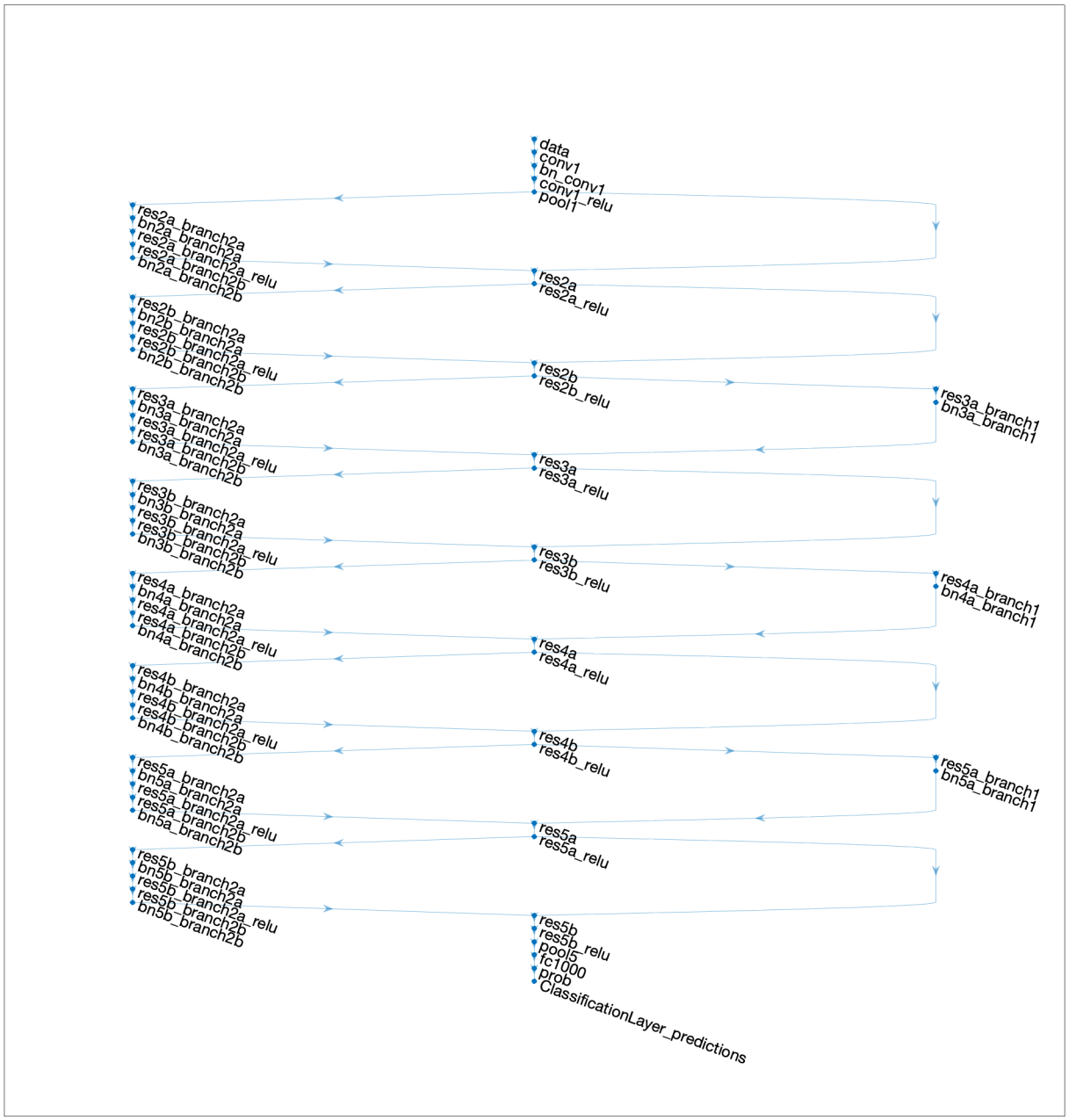
Úspěšnosti sítí byly měřeny pomocí tzv. *Top 1* a *Top 5 přesnosti*. Top 1 přesnost byla napočítávána jako podíl úspěšných klasifikací vůči celkovému počtu klasifikací, tj.

$$\text{Top 1 přesnost} = \frac{\text{\# úspěšných klasifikací}}{\text{\# klasifikací celkově}}. \tag{5.1}$$

Top 5 přesnost napočítáváme opět jako podíl úspěšných klasifikací vůči celkovému počtu klasifikací jen s tím rozdílem, že klasifikaci považujeme za úspěšnou, jakmile je správná třída mezi prvními pěti výstupy s nejvyšší pravděpodobností.

5.2 Experiment se sítí Res-Net 18

Konvoluční neuronová síť Res-Net 18 má celkem 71 vrstev a vyžaduje 45Mb paměti. V grafu 5.1 je vyobrazena struktura této sítě.



Obrázek 5.1: *Struktura konvoluční neuronové sítě Res-Net 18*

Pro úspěšný transfer learning bylo třeba znovu natrénovat poslední tři vrstvy sítě stejným způsobem jako bylo v (3.3.1). Jednalo se o vrstvy typu *FullyConnectedLayer*, *SoftmaxLayer* a *ClassificationOutputLayer*. Experimentálně bylo zjištěno, že optimální parametry pro Transfer Learning jsou:

InitialLearnRate = 0.00003,

MaxEpochs = 6,
Shuffle = *every-epoch*.

S natrénovanou sítí můžeme přejít k samotné implementaci náhrady konvolučních vrstev. V Res-Net 18 jsme postupně nahrazovali všechny konvoluční vrstvy, které měly rozměry filtrů 3×3 a 7×7. Náhradu jsme prováděli dvěmi způsoby a tím jsme vytvořili dvě strukturálně stejné, ale funkčností odlišné sítě. V jednom případě jsme k rozkladu konvolučního jádra použili Krylov-Levenberg-Marquardtův algoritmus[1] (KLM) 4.2 (viz. zdrojové kódy [17]) a v druhém případě algoritmus Nonlinear Least Squares (cpd_nls) z knihovny Tensorlab 3.0 [14]. Výsledné sítě jsme porovnali na základě přesnosti rozpoznávání Top 1 a Top 5 (5.1). Přikládáme pseudokód experimentu:

Data: net = Res-Net 18, data = $\frac{1}{4}$databáze ILSVRC 12, rozkladový algoritmus (KLM / cpd_nls), Sensitivita
Result: net1 = komprimovaná Res-Net 18
[imdsTrain, imdsTest] = splitDatabase(data, 0.7) ;
net1 = net ;
for *i = 1 : # konvolučních vrstev* **do**
  T = net1.convLayer(i).Weights ;
  $[A, B, C, D] = CPalgoritmus(T, Rank, Sensitivita)$;
  % Náhrada novými konv. vrstvami ;
  conv(i.1).Weights = C ;
  conv(i.2).Weights = A ;
  conv(i.3).Weights = B ;
  conv(i.4).Weights = D ;
  % Fine-tuning sítě ;
  net1 = trainNetwork(imdsTrain, net1);
end

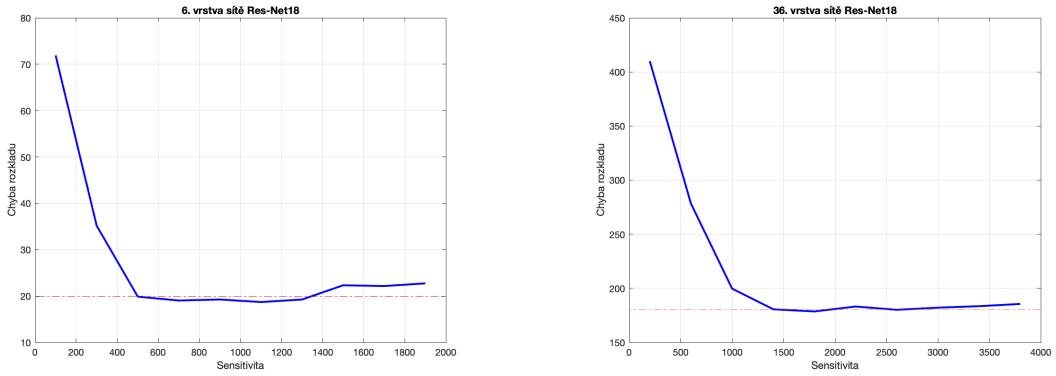
Příkaz 'trainNetwork()' značí, že v návaznosti na jednotlivé náhrady konvolučních vrstev bylo vždy prováděno doučení sítě nebo-li fine-tuning. Pro lepší přehlednost textu zde uvádíme jen princip záměny a nebudeme uvádět podrobný popis všech programovacích příkazů vystupujících v této metodě. Tyto detaily, nastavení parametrů bias a padding a nastavení parametrů fine-tuning jsou k nalezení v příloze C.

Počet iterací rozkladových algoritmů byl nastaven tak, aby jednotlivé faktorové matice stihly dokonvergovat a nebo se alespoň aproximační chyba již dále nemohla výrazněji zlepšit.

5.2.1 Určení sensitivity pro algoritmus KLM s omezením

Poslední věc, kterou je před spuštěním komprese třeba rozhodnout, je nastavení sensitivity pro algoritmus KLM. Sensitivitu bychom rádi volili tak, aby nebyla pro rozklad moc omezující a nezpůsobovala tím přílišně velkou chybu aproximace. Proto jsme provedli s jednotlivými konvolučními jádry sítě Res-Net 18 test na sensitivitu CP rozkladu. Pro každý jeden rozměr konvolučního jádra vyskytujícího se v této síti jsme vzali jeden příklad, na který jsme vícekrát aplikovali algoritmus KLM s různými hodnotami sensitivity. V průběhu jsme zapisovali chybu rozkladu, abychom na konci byli schopni určit vhodné nastavení parametru sensitivity při nahrazování konvolučních vrstev.

[1] Hodnost komplexity aproximace v Krylov-Levenberg-Marquardtovu algoritmu jsme vždy volili $M = 2$ viz. sekce 4.2.3.



(a) (b)

Obrázek 5.2: Výsledky testu sensitivity na 6. a 36. vrstvě sítě Res-Net 18 pomocí algoritmu KLM. Červená šrafovaná čára značí hladinu chyby pod kterou se rozklad nijak výrazněji nezlepšuje.

Obrázek 5.2 shrnuje výsledky testu na příkladu druhé a páté konvoluční vrstvy sítě. Je možné si všimnout, že od určité úrovně sensitivity se chyba rozkladu již výrazněji nezlepšuje. Tento zlom nastává na příkladu šesté vrstvy při sensitivitě rovné 500. Konvoluční jádro této vrstvy má rozměry $3\times3\times64\times64$ a proto na základě tohoto testu budeme při hledání CP rozkladu konvolučních jader stejné velikosti požadovat sensitivitu rovnu 500.

Analogicky na příkladu 36. vrstvy, která má rozměry $3 \times 3 \times 128 \times 256$, tak budeme při hledání rozkladů jader o stejné velikosti požadovat sensitivitu 1000.

Kompletní výsledky testu sensitivity pro všechny rozměry tenzorů objevujících se v síti Res-Net 18 lze nalézt v příloze D.

5.2.2 Výsledky komprese

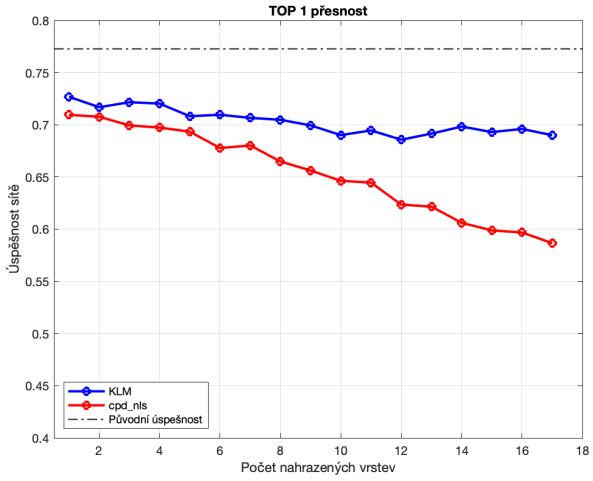
Výsledky experimentu jsou shrnuty grafy 5.3. Graf (a) zobrazuje Top 1 přesnost sítě po náhradě jednotlivých konvolučních vrstev a fine-tuningu. Je zřejmé, že síť, kde byly konvoluční vrstvy nahrazeny pomocí CP rozkladového algoritmu KLM, je výrazně úspěšnější než síť, kde byl použit obyčejný nls_cpd. Úspěšnost sítě s KLM sice postupem mírně klesala, ale v žádnou chvíli se z původních 92% nedostal pod hranici 69% Top 1 přesnosti rozpoznávání. Na druhé straně úspěšnost sítě, kde byl použit algoritmus nls_cpd, klesala výrazně rychleji a skončila na výsledných 59% úspěšnosti rozpoznávání.

Přesnost Top 5 je vyobrazena v grafu (b). Z původních 95% Top 5 úspěšnost sítě po kompresi pomocí KLM klesla na 88% zatímco po kompresi pomocí nls_cpd se úspěšnost ustálila na hodnotě 82%.

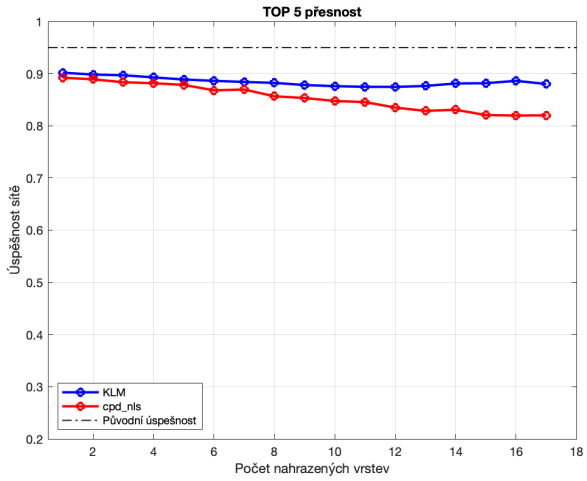
Pro lepší představu o hodnotách sensitivity jednotlivých rozkladů přikládáme graf 5.4. Graf znázorňuje sensitivitu rozkladů vah 13. konvoluční vrstvy a zároveň udává úspěšnost sítě po využití těchto rozkladů. Porovnáváme na jedné straně CP rozklad pomocí KLM a na straně druhé CP rozklad pomocí nls_cpd. Můžeme si všimnout, že nls_cpd rozklad má mnohonásobně větší sensitivitu, v tomto případě nabývá hodnoty přes 10^7, zatímco rozklad pomocí KLM splňuje podmínku omezení sensitivity na hodnotu 1400. Rozdíl můžeme zaznamenat i v úspěšnosti sítě. Na jedné straně, po implementaci rozkladu z algoritmu KLM, si síť drží úspěšnost 69% a na druhé straně po implementaci rozkladu z cpd_nls má síť úspěšnost podstatně horší, a to 62%.

Co se úspor paměti týče, tak jsme u obou sítí používali rozklady o stejné hodnosti, tudíž jsou úspory identické. Součet parametrů vah všech konvolučních vrstev původní sítě Res-Net 18 je celkově
10 994 880 objektů typu single v Matlabu, tj 43,9Mb paměti. Po nahrazení všech konvolučních vrstev

zabírají tyto konvoluční jádra nové sítě 2 613 356 parametrů typu single, což znamená 10,4Mb. Jedná se tedy o úsporu více než 75% nároků na paměť. Podrobný výčet nastavení hodností pro jednotlivé konvoluční jádra lze naleznout v příloze C.

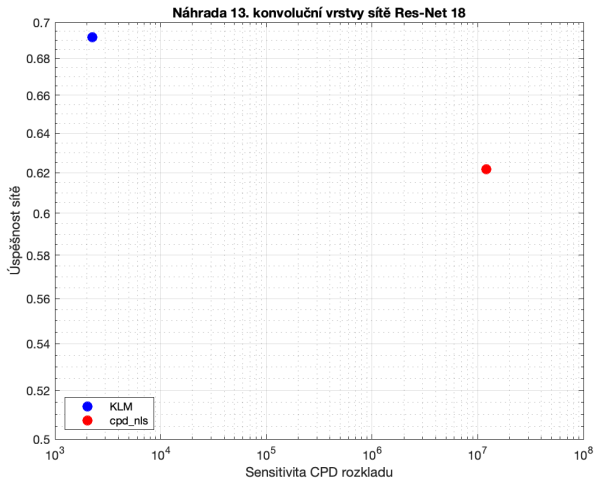


(a)



(b)

Obrázek 5.3: *Porovnání úspěšnosti sítě Res-Net 18 po kompresi konvolučních vrstev pomocí algoritmů KLM a nls_cpd.*



Obrázek 5.4: *Náhrada 13. konvoluční vrstvy sítě Res-Net 18.*

5.3 Experiment se sítí VGG-16

Konvoluční neuronová síť VGG-16 má celkem 41 vrstev a oproti Res-Net neobsahuje žádná větvení. VGG-16 vyžaduje celkem 554Mb paměti. V grafu 5.5 je vyobrazena struktura této sítě.

Se sítí VGG-16 jsme provedli identický experiment jako se sítí Res-Net 18. Opět jsme provedli Transfer Learning, kde bylo zapotřebí znovu natrénovat poslední tři vrstvy sítě, tj. vrstvy typu FullyConnectedLayer, SoftmaxLayer a ClassificationOutputLayer. Transfer learning jsme provedli s nastavením:

InitialLearnRate = 0.00003,
MaxEpochs = 6,
Shuffle = *every-epoch*.

S natrénovanou sítí jsme opět provedli test k odhadnutí sensitivity rozkladu jednotlivých konvolučních jader. Výsledky tohoto testu, podle kterých byla určena sensitivita rozkladů pro algoritmus KLM, jsou k nalezení v D.

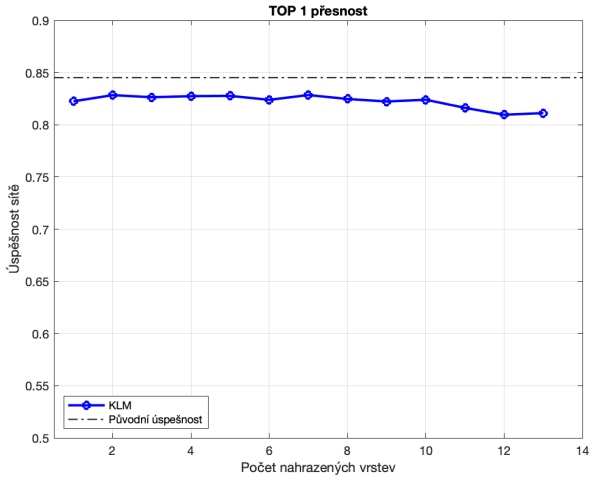
Následně jsme se sítí VGG 16 prováděli postupnou náhradu jednotlivých konvolučních vrstev pomocí algoritmu KLM.

5.3.1 Výsledky komprese

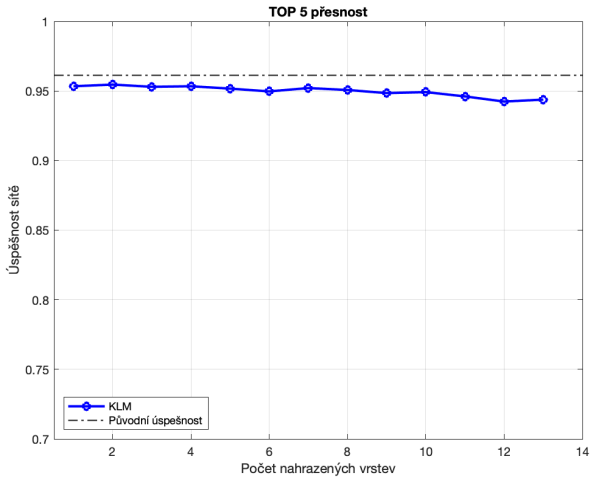
Síť VGG 16 po metodě transfer learningu dosahovala Top 1 přesnosti 84,52%. Po kompresi všech třinácti konvolučních vrstev Top 1 přesnost mírně klesla na hodnotu 81,12%. Podobný vývoj zaznamenala i Top 5 přesnost, která po transfer learningu byla na hodnotě 96,13%. Top 5 přesnost po kompresi klesla jen o necelé dva procentní body na 94,38%. Výsledky jsou opět shrnuty formou grafu 5.6 a podrobný vývoj úspěšnosti sítě v průběhu komprese lze nalézt v příloze D.

Co se úspor paměti týče, tak k uchování všech konvolučních jader původní sítě VGG 16 bylo zapotřebí 14 710 464 parametrů typu single, tj. síť vyžadovala 58,8 Mb volné paměti. Po kompresi se objem všech konvolučních vrstev snížil na 3 052 698 parametrů typu single, což znamená 12,2Mb paměti a jedná se tedy o úsporu 79% nároků na paměť při jen zanedbatelném zhoršení funkčnosti sítě.

Obrázek 5.5: *Struktura konvoluční neuronové sítě VGG-16*



(a)



(b)

Obrázek 5.6: *Vývoj úspěšnosti sítě VGG 16 v průběhu komprese konvolučních vrstev pomocí algoritmu KLM.*

Závěr

V této práci jsme na příkladu konvolučních neuronových sítí aplikovali metodu komprese konvolučních vrstev pomocí kanonického rozkladu tenzorů. V práci jsme se zaměřili na sítě určené pro problémy rozpoznávání obrazu za účelem vyvinutí optimální strategie pro jejich kompresi.

Nejprve jsme zkoumali konvoluční jádra relativně malých konvolučních sítí. V síti určené pro rozpoznávání číslic a v síti AlexNet jsme zaměnili tenzory konvolučních jader za jejich aproximaci kanonickým rozkladem a hledali jsme optimální hodnost této aproximace. Ukázalo se, že i při hodnostech, které by zaručili úsporu až 70% paměti, se chybovost takto malých sítí drží pod 5%. Tento výstup nás motivoval do hledání podobných úspor i na příkladu reálně používaných konvolučních neuronových sítí.

Hlavním výstupem této práce je komprese konvolučních neuronových sítí Res-Net 18 a VGG 16. Sítě Res-Net 18 a VGG 16 v počítači zabírají 45Mb a 554Mb. Po kompresi tyto sítě vyžadují 10.7Mb a 494.6Mb místa v paměti a množství parametrů všech konvolučních jader se sníží o více než 75%.

V průběhu komprese jsme dále ukázali, že parametr sensitivity kanonického rozkladu je zásadní při využití této metody. Komprese byla prováděna dvěmi způsoby. Jednou byl kanonický rozklad konvolučních jader hledán pomocí Krylov-Levenberg-Marquardtova algoritmu s podmínkou na omezenou sensitivitu rozkladu a v druhém případě byl rozklad hledán pomocí standardního Nonlinear Least Squares algoritmu z knihovny Tensorlab 3.0. U sítí, kde byl použit algoritmus KLM, se komprese projevila jen malým zhoršením přesnosti rozpoznávání. Oproti tomu u sítí, kde bylo použito NLS bez omezení na sensitivitu rozkladu, tak došlo k výraznějšímu zhoršení. Za opodstatnění tohoto výsledku je považována právě stabilita rozkladu vůči malým změnám, které nastávají například při doučování sítě.

Appendices

Příloha A

Náhrada konvoluční vrstvy CP aproximací

Experiment s databází DigiDataset

V tabulkách níže přikládáme numerické výsledky závislosti přesnosti rozpoznávání neuronové sítě na hodnosti rozkladu vah jejich konvolučních vrstev. Číslice v prvním sloupci značí hodnost CP rozkladu, druhý sloupec vyjadřuje Top 1 (5.1) přesnost rozpoznávání sítě a třetí sloupec sděluje chybu rozkladu. Chyba (Error) byla napočítávána jako

$$\text{Error} = \|\mathcal{X} - [[A, B, C, D]]\|_F^2.$$

2. konvoluční vrstva

Hodnost	Top 1	Error
-	0.9783	0
38	0.978	0.0062692
37	0.97867	0.0057664
36	0.97833	0.00914
35	0.978	0.012265
34	0.978	0.017103
33	0.97933	0.017661
32	0.978	0.030648
31	0.97767	0.03045
30	0.97667	0.039738
29	0.97733	0.047549
28	0.97633	0.051828
27	0.97767	0.062218
26	0.97267	0.073112
25	0.97967	0.082374
24	0.976	0.094356
23	0.97267	0.10211
22	0.96367	0.12099
21	0.96867	0.14068
20	0.97167	0.16169
19	0.96367	0.18013
18	0.95867	0.19787
17	0.96433	0.22783
16	0.956	0.24609
15	0.93867	0.28387
14	0.944	0.30951
13	0.902	0.3653
12	0.89033	0.39757
11	0.832	0.4552
10	0.901	0.50205
9	0.839	0.56158
8	0.63767	0.62153
7	0.45933	0.69565
6	0.453	0.78799
5	0.48667	0.87085
4	0.17433	0.98698
3	0.34767	1.1114
2	0.12933	1.2129
1	0.10067	1.348

3. konvoluční vrstva

Hodnost	Top 1	Error
-	0.9783	0
85	0,9783	0,0024
84	0,9786	0,0037
83	0,977	0,0039
82	0.97733	0.004315
81	0.97933	0.006105
80	0.97867	0.0073311
79	0.979	0.0087966
78	0.97833	0.01135
77	0.978	0.0134
76	0.978	0.012853
75	0.97833	0.016839
74	0.97867	0.015731
73	0.97733	0.020032
72	0.976	0.027388
71	0.97733	0.026399
70	0.97733	0.035025
69	0.978	0.038284
68	0.97767	0.034555
67	0.97733	0.040726
66	0.97767	0.043784
65	0.977	0.053656
64	0.97867	0.051949
63	0.97567	0.057867
62	0.97567	0.068051
61	0.976	0.068567
60	0.97767	0.077906
59	0.976	0.085531
58	0.97667	0.08588
57	0.97533	0.094769
56	0.97533	0.10527
55	0.97467	0.11378
54	0.974	0.11924
53	0.973	0.12163
52	0.97267	0.1324
51	0.97267	0.14621
50	0.97267	0.1475
49	0.97167	0.16268
48	0.97467	0.17137
47	0.974	0.17867
46	0.97233	0.18954
45	0.96933	0.20187
44	0.97267	0.20608
43	0.97067	0.21807
42	0.96867	0.2303
41	0.972	0.24449
40	0.96667	0.25461
39	0.96867	0.27695
38	0.97267	0.28132
37	0.97067	0.29833
36	0.96733	0.31961
35	0.96567	0.32417
34	0.96367	0.34297
33	0.964	0.36747
32	0.96	0.37602
31	0.96333	0.40117
30	0.95767	0.41758
29	0.95567	0.43049
28	0.95133	0.45956
27	0.95	0.47125
26	0.954	0.49951
25	0.94033	0.51487
24	0.94267	0.54951
23	0.941	0.58436
22	0.93267	0.60804
21	0.94233	0.63067
20	0.94233	0.65755
19	0.92167	0.70279
18	0.93967	0.7272
17	0.93433	0.75487
16	0.892	0.79516
15	0.92833	0.82832
14	0.87233	0.89361
13	0.90467	0.92159
12	0.86033	0.96766
11	0.85367	1.0194
10	0.65767	1.0758
9	0.67633	1.1556
8	0.568	1.1989
7	0.46	1.2801
6	0.455	1.3802
5	0.38533	1.5006
4	0.316	1.594
3	0.20433	1.7412
2	0.23767	1.8247
1	0.18933	1.9786

Experiment se sítí AlexNet

Numerické výsledky závislosti přesnosti rozpoznávání neuronové sítě na hodnosti rozkladu vah jejich konvolučních vrstev. Číslice v prvním sloupci značí hodnost CP rozkladu, druhý sloupec vyjadřuje Top 1 (5.1) přesnost rozpoznávání sítě a třetí sloupec sděluje chybu rozkladu. Chyba (Error) byla napočítávána jako

$$\text{Error} = \|\mathcal{X} - [[A, B, C, D]]\|_F^2.$$

1. konvoluční vrstva neuronové sítě AlexNet

Hodnost	Top 1	Error
-	0.95989	0
80	0.96028	0.89447
79	0.95951	0.94819
78	0.95951	0.97596
77	0.95989	1.0197
76	0.95951	1.0528
75	0.95951	1.0766
74	0.95989	1.1351
73	0.95989	1.1807
72	0.95989	1.2325
71	0.96066	1.2578
70	0.96066	1.4322
69	0.95989	1.3939
68	0.96066	1.5173
67	0.95951	1.5701
66	0.95837	1.6513
65	0.95951	1.7097
64	0.95989	1.8878
63	0.95951	1.8266
62	0.95913	1.9795
61	0.95951	2.1027
60	0.95913	2.2666
59	0.96104	2.3911
58	0.95989	2.4359
57	0.95837	2.5752
56	0.95989	2.6407
55	0.96028	2.8729
54	0.95837	3.1194
53	0.95837	3.27
52	0.95913	3.414
51	0.95722	3.5185
50	0.9576	4.0239
49	0.95798	4.1157
48	0.95951	4.457
47	0.95569	4.6084
46	0.95646	5.1019
45	0.95837	5.6119
44	0.95225	5.6429
43	0.95187	6.1386
42	0.9301	6.5701
41	0.94843	7.0757
40	0.94232	7.615
39	0.94194	7.4779
38	0.94958	8.6117
37	0.94118	8.6602
36	0.93927	9.631
35	0.92781	10.934
34	0.92437	10.978
33	0.9259	11.896
32	0.92934	13.036
31	0.92475	14.748
30	0.90298	16.039
29	0.911	16.604
28	0.65699	18.423
27	0.88121	19.071
26	0.86707	20.558
25	0.74752	22.106
24	0.65508	24.796
23	0.7796	25.828
22	0.73568	27.817
21	0.73338	29.769
20	0.73644	32.666
19	0.49885	36.368
18	0.534	37.347
17	0.56684	40.237
16	0.48625	43.707
15	0.56837	46.717
14	0.46333	49.993
13	0.40756	53.509
12	0.34072	56.744
11	0.20779	60.663
10	0.18678	64.634
9	0.2013	69.816
8	0.14133	75.075
7	0.1631	79.487
6	0.10122	84.661
5	0.09893	89.082
4	0.065699	97.104
3	0.05806	101.93
2	0.024446	108.41
1	0.049656	112.73

Příloha B

Vzorce pro výpočet Jakobiho matice

Výpočet Jakobiho matice

Nechť máme tenzor $\mathcal{T}$, k němuž hledáme kanonický rozklad $[[A, B, C, D]]$ o hodnosti R. Budeme se držet značení z 4.2.4, kde

$$\theta_A = \mathrm{vec}(A), \;\; \theta_B = \mathrm{vec}(B), \;\; \theta_C = \mathrm{vec}(C), \;\; \theta_D = \mathrm{vec}(D),$$
$$\theta = [\theta_A^T, \theta_B^T, \theta_C^T, \theta_D^T]^T,$$
$$\mathcal{T}(\theta) = [[A, B, C, D]].$$

Jakobiho matici můžeme zapsat jako

$$J = [J_A, J_B, J_C, J_D] = \left[\frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}A}, \frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}B}, \frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}C}, \frac{\partial \mathcal{T}(\theta)}{\partial \mathrm{vec}D}\right].$$

Pro lepší přehlednost odvození výpočtu jednotlivých částí Jakobiho matice si nejprve rozepíšeme i-tý element výrazu $\mathrm{vec}\mathcal{T}(\theta)$, tj.

$$(\mathrm{vec}\mathcal{T}(\theta))_i = \left(\mathrm{vec}A\,(D \odot (C \odot B))^T\right)_i = ((D \odot C \odot B \odot A)1_R)_i\,.$$

Tento výraz můžeme dále přepisovat do tvarů, které umožní snadné odvození derivace podle $\theta_A, \theta_B, \theta_C$ a θ_D.

$$(D \odot C \odot B \odot A)1_R = \begin{cases} ((D \odot C \odot B) \otimes \mathbb{I}_I)\theta_A\,, \\ \{D_{:,i} \otimes C_{:,i} \otimes (\mathbb{I}_J \otimes A_{:,i})\}_{i \in R}\theta_B\,, \\ \{D_{:,i} \otimes (\mathbb{I}_K \otimes (B_{:,i} \otimes A_{:,i}))\}_{i \in R}\theta_C\,, \\ \{\mathbb{I}_L \otimes (C_{:,i} \otimes B_{:,i} \otimes A_{:,i})\}_{i \in R}\theta_D\,. \end{cases} \tag{B.1}$$

A tím dostáváme

$$\begin{aligned} \frac{\partial \mathrm{vec}\mathcal{T}(\theta)}{\partial \theta_A} &= (D \odot C \odot B) \otimes \mathbb{I}_I, \\ \frac{\partial \mathrm{vec}\mathcal{T}(\theta)}{\partial \theta_B} &= \{D_{:,i} \otimes C_{:,i} \otimes (\mathbb{I}_J \otimes A_{:,i})\}_{i \in R}, \\ \frac{\partial \mathrm{vec}\mathcal{T}(\theta)}{\partial \theta_C} &= \{D_{:,i} \otimes \mathbb{I}_K \otimes (B_{:,i} \otimes A_{:,i})\}_{i \in R}, \\ \frac{\partial \mathrm{vec}\mathcal{T}(\theta)}{\partial \theta_D} &= \{\mathbb{I}_L \otimes (C_{:,i} \otimes B_{:,i} \otimes A_{:,i})\}_{i \in R}. \end{aligned} \tag{B.2}$$

Příloha C

Podrobnosti implementace

Pravidla pro volbu tlumícího parametru μ

Pravidla pro volbu parametru μ byla převzata z [25]. K určení parametru μ je navíc zapotřebí pomocný parametr ρ. Inicializace obou dvou parametrů probíhá následovně. Mějme faktorové matice $[[A, B, C, D]]$. Nejprve si označíme,

$$\text{na} = \left\{\sum_{i=1}^{\text{Ia}} A(i,r)^2\right\}_{r=1}^{R} \quad \text{nb} = \left\{\sum_{i=1}^{\text{Ib}} B(i,r)^2\right\}_{r=1}^{R} \quad \text{nc} = \left\{\sum_{i=1}^{\text{Ic}} C(i,r)^2\right\}_{r=1}^{R} \quad \text{nd} = \left\{\sum_{i=1}^{\text{Id}} D(i,r)^2\right\}_{r=1}^{R}$$

a následně inicializujeme

$$\rho_0 = 2, \qquad \mu_0 = \max\{\text{nb}\cdot\text{nc}\cdot\text{nd}, \text{na}\cdot\text{nc}\cdot\text{nd}, \text{na}\cdot\text{nb}\cdot\text{nd}, \text{na}\cdot\text{nb}\cdot\text{nc}\}.$$

Určení ρ_{k+1} a μ_{k+1} rozdělíme na dva případy. V prvním případě, pokud účelová funkce v k+1-ní iteraci neklesla, tak

$$\mu_{k+1} = \rho_k \mu_k, \qquad \rho_{k+1} = 2\rho_k.$$

V opačném případě, pokud účelová funkce klesla,

$$\rho = \frac{f(\theta_k) - f(\theta_{k+1})}{d^T(H + \mu d)}, \quad \text{kde } d = (H + \mu\mathbb{I})^{-1} g,$$

$$\mu_{k+1} = \mu_k \max\{\frac{1}{3}, 1 - (2\rho - 1)^3\}, \quad \rho_{k+1} = 2.$$

Komprese konvoluční vrstvy v prostředí MATLAB

Popis příkazů v prostředí Matlab

Mějme CP rozklad $\mathcal{T} = [[A, B, C, D]]$, kde $\mathcal{T}$ je konvoluční jádro původní konvoluční vrstvy. MATLAB neumožňuje změny parametrů objektů typu DAGNetwork, jehož typu je v Matlabu většina neuronových sítí a proto je třeba nejprve síť převést na objekt typu *LayerGraph* pomocí příkazu

$$tmp\_net = layerGraph(net);$$

kde *net* je objekt typu DAGNetwork, pro který chceme záměnu vrstvy provádět. Nyní potřebujeme vytvořit čtyři nové konvoluční vrstvy. První konvoluční vrstva bude mít filtry o rozměrech $1 \times R$. Hodnotu

Padding zde nastavíme rovnu nule a stejně tak i bias nastavíme samé nuly. Aby při následném učení nedocházelo k adaptaci prahu v této vrstvě, tak nastavíme faktor *BiasLearnRateFactor* na nulu. Za váhy této konvoluční vrstvy položíme faktorovou matici C.

$$\begin{aligned} conv\_1 \;=\; & convolution2dLayer(1, R, \text{`}Padding\text{`}, 0, \text{`}BiasLearnRateFactor\text{`}, 0, ... \\ & \text{`}Weights\text{`}, single(reshape(C, 1, 1, [], R)), \text{`}Bias\text{`}, zeros(1, 1, R)); \end{aligned}$$

Aby náhrada odpovídala odvození v 3.1, tak další dvě konvoluční vrstvy je třeba zvolit typu *groupedConvolution2dLayer()* a způsob konvoluce nastavit na *channel-wise*. Do těchto dvou vrstev přeneseme padding. Práh bude nastaven na 0 a bude opět zamezeno jeho dalšímu učení. Za váhy těchto dvou vrstev položíme faktorové matice A a B.

$$\begin{aligned} conv\_2 \;=\; & convolution2dLayer([size(T, 1), 1], 1, \text{`}channel - wise\text{`}, ... \\ & \text{`}Padding\text{`}, [Size(Padding, 1), 0], \text{`}BiasLearnRateFactor\text{`}, 0, ... \\ & \text{`}Weights\text{`}, single(reshape(A, [], 1, 1, 1, R)), \text{`}Bias\text{`}, zeros(1, 1, 1, R)); \end{aligned}$$

$$\begin{aligned} conv\_3 \;=\; & convolution2dLayer([size(T, 2), 1], 1, \text{`}channel - wise\text{`}, ... \\ & \text{`}Padding\text{`}, [Size(Padding, 3), 0], \text{`}BiasLearnRateFactor\text{`}, 0, ... \\ & \text{`}Weights\text{`}, single(reshape(B, 1, [], 1, 1, R)), \text{`}Bias\text{`}, zeros(1, 1, 1, R)); \end{aligned}$$

Čtvrtá konvoluční vrstva bude mít filtry o rozměrech 1×1 a navíc převezme bias z původní konvoluční vrstvy. Za váhy položíme matici D.

$$\begin{aligned} conv\_4 \;=\; & convolution2dLayer([1, 1], NumberOutChannels, \text{`}Padding\text{`}, 0, ... \\ & \text{`}Weights\text{`}, single(reshape(D', 1, 1, R, [], 1)), \text{`}Bias\text{`}, reshape(prevBias, 1, 1, []), ... \\ & \text{`}NumChannels\text{`}, R); \end{aligned}$$

Vrstvy jsou vytvořené a je třeba je správně napojit do původní sítě. Zde nastává problém se sítí ResNet 18. Jedná se totiž o residuální síť, tj. v určitých vrstvách se větví a tudíž neplatí pravidlo, že předešlá vrstva v seznamu tmp_net.Layers přímo navazuje na vrstvu po ní následující. Úplný přehled propojení jednotlivých vrstev sítě můžeme nalézt v záložce *net.Connections*. V té jednotlivá propojení vyhledáváme, abychom věděli, které vrstvy je třeba rozpojit a kam napojit vrstvy nové. Vyhledávání provádíme příkazem

$$find(strcmp(old\_name, [net.Connections])); .$$

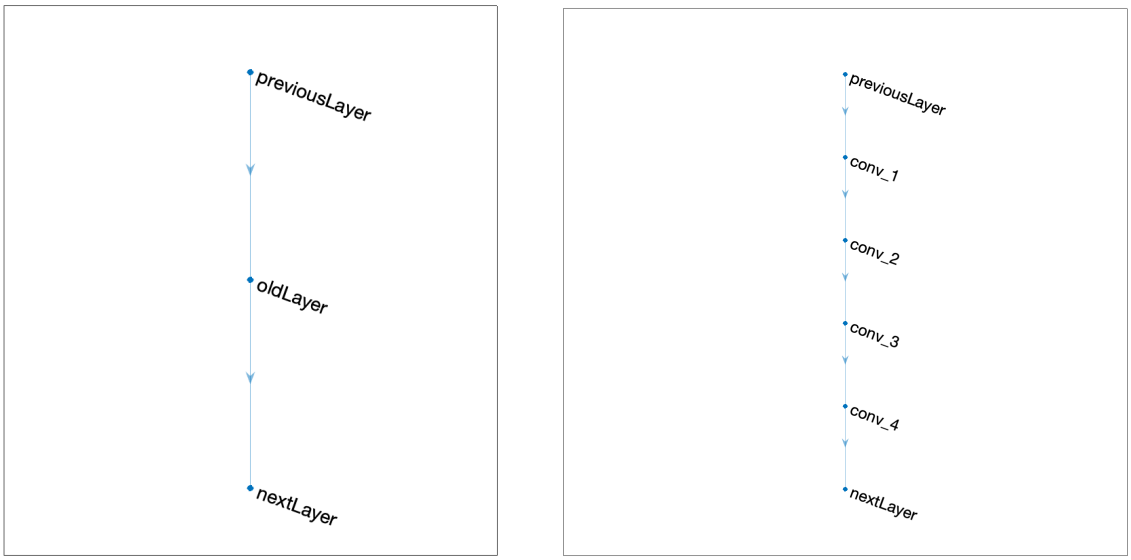
Fine-tuning

Nastavení fine-tuning zůstává pro všechny experimenty stejné a vypadá následovně

$$\begin{aligned} options \;=\; & trainingOptions(\text{`}sgdm\text{`}, ... \\ & \text{`}MiniBatchSize\text{`}, 10, ... \\ & \text{`}InitialLearnRate\text{`}, 3e-3, ... \\ & \text{`}Shuffle\text{`}, \text{`}every - epoch\text{`}); \end{aligned}$$

K samotnému dotrénování je potřeba následně zavolat příkaz

$$net = trainNetwork(trainData, tmp\_net, options); .$$



(a) (b)

Obrázek C.1: Původní úsek sítě vlevo. Úsek sítě po náhradě vpravo.

CP rozklad jednotlivých vrstev sítě Res-Net 18

Následuje tabulka shrnující rozměry konvolučních jader jednotlivých konvolučních vrstev a hodnost, která jim byla určena.

Konvoluční vrstva	Rozměr jádra	Hodnost
1	$7 \times 7 \times 3 \times 64$	60
2	$3 \times 3 \times 64 \times 64$	76
3	$3 \times 3 \times 64 \times 64$	76
4	$3 \times 3 \times 64 \times 64$	76
5	$3 \times 3 \times 64 \times 64$	76
6	$3 \times 3 \times 64 \times 128$	92
7	$3 \times 3 \times 128 \times 128$	126
8	$3 \times 3 \times 128 \times 128$	126
9	$3 \times 3 \times 128 \times 128$	126
10	$3 \times 3 \times 128 \times 256$	164
11	$3 \times 3 \times 256 \times 256$	240
12	$3 \times 3 \times 256 \times 256$	240
13	$3 \times 3 \times 256 \times 256$	240
14	$3 \times 3 \times 256 \times 512$	320
15	$3 \times 3 \times 512 \times 512$	572
16	$3 \times 3 \times 512 \times 512$	572
17	$3 \times 3 \times 512 \times 512$	572

CP rozklad jednotlivých vrstev sítě VGG 16

Následuje tabulka shrnující rozměry konvolučních jader jednotlivých konvolučních vrstev a hodnost, která jim byla určena.

Konvoluční vrstva	Rozměr jádra	Hodnost
1	$3 \times 3 \times 3 \times 64$	60
2	$3 \times 3 \times 64 \times 64$	76
3	$3 \times 3 \times 64 \times 128$	92
4	$3 \times 3 \times 128 \times 128$	126
5	$3 \times 3 \times 128 \times 256$	164
6	$3 \times 3 \times 256 \times 256$	240
7	$3 \times 3 \times 256 \times 256$	240
8	$3 \times 3 \times 256 \times 512$	320
9	$3 \times 3 \times 512 \times 512$	572
10	$3 \times 3 \times 512 \times 512$	572
11	$3 \times 3 \times 512 \times 512$	572
12	$3 \times 3 \times 512 \times 512$	572
13	$3 \times 3 \times 512 \times 512$	572

Příloha D

Výsledky

Test sensitivity konvolučních jader

Pro každý rozměr bylo ze sítě vybrán jeden reprezentant, na kterého byl postupně aplikován algoritmus KLM s postupně se zvyšujícím parametrem sensitivity. Chyba byla napočítávána jako suma čtverců matrizovaného tenzoru $\mathcal{X}$ a příslušného produktu výsledných faktorových matic $[[A, B, C, D]]$, tj.

$$\text{Error} = \|\mathcal{X} - [[A, B, C, D]]\|_F^2.$$

Algoritmus KLM byl nastaven na 1000 iterací a výsledky shrnujeme v následujících tabulkách. Sensitivita, která poté byla použita v experimentu náhrad konvolučních vrstev, je v tabulkách vyznačena červenou barvou. Stejný experiment byl aplikován jak na síť Res-Net 18 tak na síť VGG 16.

$7 \times 7 \times 3 \times 64$

Sensitivita	Error
100	111.0260
300	22.0363
500	4.4649
700	5.1763
900	3.0545
1100	3.2265
1300	3.3937
1500	3.2785
1700	3.1098
1900	3.5651

$3 \times 3 \times 64 \times 64$

Sensitivita	Error
100	71.9067
300	35.1570
500	19.8719
700	19.0318
900	19.2575
1100	18.7032
1300	19.2508
1500	22.3215
1700	22.1560
1900	22.7348

$3 \times 3 \times 64 \times 128$

Sensitivita	Error
100	192.0391
300	120.9453
500	86.2728
700	74.2265
900	68.3259
1100	68.9939
1300	70.0173
1500	70.8257
1700	70.6265
1900	69.4132

$3 \times 3 \times 128 \times 128$

Sensitivita	Error
100	296.4937
300	212.0437
500	168.5141
700	142.0951
900	134.7574
1100	132.3295
1300	133.1660
1500	132.9557
1700	133.0615
1900	131.3837

$3 \times 3 \times 128 \times 256$

Sensitivita	Error
200	409.9925
600	278.2998
1000	199.7586
1400	180.6036
1800	178.6004
2200	183.1596
2600	180.1512
3000	182.1169
3400	183.4821
3800	185.5495

$3 \times 3 \times 256 \times 256$

Sensitivita	Error
200	666.0842
600	504.1677
1000	422.2846
1400	324.6127
1800	309.6627
2200	301.6183
2600	295.0662
3000	292.7261
3400	290.7447
3800	292.6301

$3 \times 3 \times 256 \times 512$

Sensitivita	Error
250	938.4910
750	740.1503
1250	616.3018
1750	530.1332
2250	485.5928
2750	451.7504
3250	421.7526
3750	453.3854
4250	433.6615
4750	445.1982

$3 \times 3 \times 512 \times 512$

Sensitivita	Error
300	1412.8270
900	1179.7120
1500	971.9266
2100	845.4099
2700	769.3965
3300	761.8107
3900	739.0550
4500	716.6266
5100	691.2278
5700	693.4838

Test sensitivity konvolučních jader sítě VGG 16

$3 \times 3 \times 3 \times 64$

Sensitivita	Error
10	27.7289
30	5.5372
50	0.6771
70	0.0832
90	0.0263
110	0.0621
130	0.0483
150	0.0251
170	0.0155
190	0.0193

$3 \times 3 \times 64 \times 64$

Sensitivita	Error
100	6.3582
300	1.4236
500	2.0243
700	2.4917
900	2.4513
1100	3.3244
1300	3.8804
1500	2.8339
1700	5.0327
1900	3.2732

$3 \times 3 \times 64 \times 128$

Sensitivita	Error
100	18.6073
300	3.0212
500	3.2451
700	3.9123
900	5.5705
1100	6.2227
1300	11.6084
1500	9.1556
1700	10.5895
1900	13.0508

$3 \times 3 \times 128 \times 128$

Sensitivita	Error
100	27.9841
300	11.0049
500	7.4747
700	8.5203
900	9.5231
1100	13.2199
1300	13.4814
1500	16.6177
1700	17.3118
1900	18.7553

$3 \times 3 \times 128 \times 256$

Sensitivita	Error
200	16.4717
600	9.2092
1000	11.9976
1400	14.5833
1800	22.3862
2200	29.1729
2600	27.0645
3000	26.7468
3400	60.4596
3800	52.2781

$3 \times 3 \times 256 \times 256$

Sensitivita	Error
200	26.0043
600	12.3904
1000	14.3765
1400	13.6360
1800	19.1540
2200	21.8810
2600	37.6247
3000	57.4539
3400	39.0288
3800	63.3276

$3 \times 3 \times 256 \times 512$

Sensitivita	Error
250	38.4751
750	20.5841
1250	23.0533
1750	27.4279
2250	32.0218
2750	32.9249
3250	49.4582
3750	61.0740
4250	51.7583
4750	56.8740

$3 \times 3 \times 512 \times 512$

Sensitivita	Error
300	52.6826
900	33.7068
1500	37.4383
2100	43.6784
2700	44.9129
3300	59.3009
3900	65.1700
4500	74.3245
5100	73.3655
5700	115.1842

Výsledky náhrady konvolučních vrstev sítě Res-Net 18 pomocí algoritmu KLM

Výsledky komprese jednotlivých konvolučních vrstev sítě Res-Net 18 pomocí rozkladového algoritmu KLM. Chyba (Error) rozkladu [[A,B,C,D]] je udávaná jako relativní chyba vůči velikosti rozklá-

daného tenzoru $\mathcal{X} \in \mathbb{R}^{I \times J \times K \times L}$, tj.

$$\text{Error} = \frac{\|\mathcal{X} - [[A, B, C, D]]\|_F^2}{I \cdot J \cdot K \cdot L}.$$

Úspěšnost Top 1 byla počítána jako podíl úspěšných klasifikací vůči celkovému počtu klasifikací

$$\text{Top 1 Accuracy} = \frac{\text{\# úspěšných klasifikací}}{\text{\# klasifikací celkově}}.$$

Úspěšnost Top 5 byla počítána stejně jako Top 1, ale za úspěšnou jsme klasifikaci považovali kdykoliv, kdy správná třída byla mezi prvním pěti třídami výstupu neuronové sítě podle pravděpodobnosti.

Úspěšnost sítě Top 1 před kompresí: 77.27%
Úspěšnost sítě Top 5 před kompresí: 92.08%

Číslo konvoluční vrstvy	Error	Top 1 Accuracy	Top 5 Accuracy
1	4.19×10^{-4}	0.7268	0.9016
2	5.13×10^{-4}	0.7168	0.8981
3	6.05×10^{-4}	0.7216	0.8967
4	8.77×10^{-4}	0.7203	0.8927
5	1.00×10^{-3}	0.7082	0.8887
6	1.60×10^{-3}	0.7096	0.8861
7	1.40×10^{-3}	0.7066	0.8840
8	9.98×10^{-4}	0.7048	0.8822
9	8.77×10^{-4}	0.6994	0.8781
10	1.30×10^{-3}	0.6901	0.8759
11	1.30×10^{-3}	0.6946	0.8747
12	5.73×10^{-4}	0.6857	0.8745
13	6.47×10^{-4}	0.6915	0.8763
14	1.20×10^{-3}	0.6983	0.8811
15	1.10×10^{-3}	0.6930	0.8816
16	8.84×10^{-4}	0.6960	0.8861
17	9.56×10^{-4}	0.6900	0.8800

Výsledky náhrady konvolučních vrstev sítě Res-Net 18 pomocí algoritmu nls_cpd

Číslo konvoluční vrstvy	Error	Top 1 Accuracy	Top 5 Accuracy
1	6.44×10^{-5}	0.7095	0.8921
2	4.08×10^{-4}	0.7077	0.8891
3	4.15×10^{-4}	0.6994	0.8834
4	6.55×10^{-4}	0.6973	0.8815
5	8.22×10^{-4}	0.6934	0.8783
6	8.25×10^{-4}	0.6777	0.8679
7	9.14×10^{-4}	0.6802	0.8695
8	8.08×10^{-4}	0.6648	0.8565
9	8.54×10^{-4}	0.6562	0.8532
10	8.61×10^{-4}	0.6463	0.8475
11	7.74×10^{-4}	0.6446	0.8454
12	5.56×10^{-4}	0.6235	0.8349
13	5.86×10^{-4}	0.6216	0.8285
14	9.40×10^{-4}	0.6061	0.8307
15	1.10×10^{-4}	0.5988	0.8205
16	8.84×10^{-4}	0.5968	0.8196
17	9.56×10^{-4}	0.5866	0.8210

Výsledky náhrady konvolučních vrstev sítě VGG 16 pomocí algoritmu KLM

Výsledky komprese jednotlivých konvolučních vrstev sítě VGG 16 pomocí rozkladového algoritmu KLM. Chyba (Error) rozkladu [[A,B,C,D]], úspěšnost sítě Top 1 a Top 5 jsou napočítávány stejně jako v experimentu se sítí Res-Net 18.

Úspěšnost sítě Top 1 před kompresí: 84.52%
Úspěšnost sítě Top 5 před kompresí: 96.13%

Číslo konvoluční vrstvy	Error	Top 1 Accuracy	Top 5 Accuracy
1	8.2×10^{-3}	0.8225	0.9534
2	2.65×10^{-5}	0.8284	0.9546
3	3.42×10^{-5}	0.8264	0.9530
4	4.49×10^{-5}	0.8274	0.9534
5	2.66×10^{-5}	0.8277	0.9517
6	1.69×10^{-5}	0.8239	0.9497
7	2.26×10^{-5}	0.8285	0.9521
8	1.41×10^{-5}	0.8247	0.9507
9	1.02×10^{-5}	0.8223	0.9485
10	1.56×10^{-5}	0.8240	0.9492
11	1.97×10^{-5}	0.8162	0.9461
12	1.79×10^{-5}	0.8096	0.9424
13	1.61×10^{-5}	0.8112	0.9438

Příloha E

Databáze ILSVRC12

Příklady obrázků z databáze ILSVRC12

Přikládáme čtyři příklady obrázků z databáze ILSVRC12. Pro kategorie "akordeon", "balón", "banjo"a "bota"přikládáme vždy po jednom reprezentantovi.

Obrázek E.1: akordeon

Obrázek E.2: balón

Obrázek E.3: banjo

Obrázek E.4: bota

Literatura

[1] T. G. Kolda, B. W. Bader: *Tensor Decomposition and Applications*. SIAM Review, vol. 51, no. 3, p. 455500, 2009.

[2] E. Volná: *Neuronové sítě 1. Skriptum*, Ostravská univerzita v Ostrave. http://www1.osu.cz/ volna/Neuronove_site_skripta.pdf, 2008.

[3] F. Hakl, M. Holeňa: *Úvod do teorie neuronových sítí*, ČVUT Praha, Fakulta jaderná a fyzikálně inženýrská, skriptum, 1997.

[4] J. Šíma, J. Neruda: *Teoretické otázky neuronových sítí*, Matfyzpress, Praha 1996.

[5] I. Goodfellow, Y. Bengio, A. Courville: *Back-Propagation and Other Differentiation Algorithms*. Deep Learning. MIT Press. pp. 200–220. ISBN 9780262035613, 2016.

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton : *ImageNet Classification with Deep Convolutional Neural Networka*, NIPS, 2012.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel : *Backpropagation Applied to Handwritten Zip Code Recognition*, Neural Computation, 1989.

[8] W. S. McCulloch, W. Pitts: *A logical calculus of the ideas immanent in nervous activity*, University of Illinois, College of medicin. Bulletin of mathematical biophysics, vol. 5, 1943.

[9] M. V. Valueva, N. N. Nagornov, P. A. Lyakhov, G. V. Valuev, N. I. Chervyakov (2020): *Application of the residue number system to reduce hardware costs of the convolutional neural network implementation*. Mathematics and Computers in Simulation vol.177, p:232-243, November 2020.

[10] N. D. Sidiropoulos: *Tensor Decomposition for Signal Processing and Machine Learning*. IEEE Transactions on Signal Processing, vol.65, no. 13, July 1, 2017.

[11] T. Kováč: *Kanonický rozklad tenzorů maticového násobení*. ČVUT v Praze, Fakulta jaderná a fyzikálně inženýrská, bakalářská práce, 2018.

[12] Mathworks: *Create Simple Deep Learning Network for Classification*, https://www.mathworks.com/help/deeplearning/examples/create-simple-deep-learning-network-for-classification.html

[13] *Matlab - Deep Learning Toolbox*, https://www.mathworks.com/products/deep-learning.html

[14] N. Vervliet, O. Debals, L. Sorbet, M. Van Barel, L. De Lathauwer: *Tensorlab 3.0*, March 2016. URL: https://www.tensorlab.net/

[15] Dataset *Caltech 101*, http://www.vision.caltech.edu/Image_Datasets/Caltech101/

[16] P. Tichavský, A. Phan, A. Cichocki : *Krylov-Levenberg-Marquardt Algorithm for Structured Tucker Tensor Decomposition*, článek v recenzním řízení.

[17] P. Tichavský : *Krylov-Levenberg-Marquardt method for CP tensor decomposition*, https://github.com/Tichavsky/tensor-decomposition/blob/master/KLM4cS.m , 2019.

[18] P. Tichavský, A.H. Phan, A Cichocki: *Partitioned Alternating Least Squares Technique for Canonical Polyadic Tensor Decomposition*, IEEE Signal Processing Letters, vol. 23, no.7, July 2016.

[19] P. Tichavský, A. Phan, A. Cichocki : *Sensitivity in tensor decomposition*, IEEE Signal Processing Letters vol.26, p. 1653-1657, November 2019.

[20] P. Tichavský, A. H. Phan, A. Cichocki : *Numerical CP Decomposition of Some Difficult Tensors* , Journal of Computational and Applied Mathematics vol.317, p. 362-370, 2017.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei : *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision 115, 211-252, 2015.

[22] Princeton University : *WordNet - A Lexical Database for English*, https://wordnet.princeton.edu/ .

[23] Researchgate.net : *Artifiacial neural network architecture figure*, https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN- i-h-1-h-2-h-n-o_fig1_321259051.

[24] *Convolutional neural network diagram*, https://diagram.alimb.us/50-draw-neural-network-diagram-online-lh1t/draw-neural-network-diagram-online-tikz-pgf-how-can-i-add-quotdotsquot-between-the-nodes-of-my/ .

[25] K. Madsen, H. B. Nielsen, O. Tingleff *Methods for nonlinear least squares problems, second ed.*, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 2004.